

DOCTORAL THESIS

---

# ARTIFICIAL INTELLIGENCE FOR AUTOMATED DECISION-MAKING IN ERROR-PATTERN RECOGNITION

---

*Author:*

**Felip GUIMERÀ CUEVAS**

*Supervisors LMU:*

**PD Dr. Helmut SCHMID**

PD Dr. Stefan LANGER

*Supervisors BMW:*

Dr. Iryna BASTIAN

Dr. Raphael WEINGARTNER



CENTER FOR INFORMATION AND LANGUAGE PROCESSING  
**LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN**

*In collaborative partnership with:*

**BMW GROUP**

EXAMINATION DATE: JULY 10, 2024

*Submitted in fulfillment of the requirements for the degree of Doctor of Philosophy.*



---

# **ARTIFICIAL INTELLIGENCE FOR AUTOMATED DECISION-MAKING IN ERROR-PATTERN RECOGNITION**

---

## **INAUGURAL DISSERTATION**

for the attainment of the Doctorate in Philosophy of the  
Ludwig-Maximilians-Universität München

*submitted by*  
**Felip GUIMERÀ CUEVAS**  
*from*  
**Barcelona (Catalonia-Spain)**

**2024**

*Academic supervisor: PD Dr. Helmut SCHMID*  
*Academic co-supervisor: PD Dr. Stefan LANGER*

---

*Date of the oral examination: July 10, 2024*

# Declaration of Authorship

I, Felip GUIMERÀ CUEVAS, declare that this thesis titled, "Artificial Intelligence for Automated Decision-Making in Error-Pattern Recognition", and the work presented in it are my own and have originated as a result of my own original research.

## I affirm the following:

- This thesis comprises original content that has neither been previously submitted for a degree or any other qualification at this University or any other institution. In cases where specific sections incorporate or build upon any previously submitted work, appropriate acknowledgment is provided to indicate the connection with prior contributions.
- Whenever I have consulted the published or unpublished work of others, I have diligently attributed them, providing accurate and complete references.
- Except for duly acknowledged quotations and references, this thesis represents the result of the entirety of my own research. In instances where the thesis is based on collaborative work with others, I have provided transparent acknowledgment and referencing of their contribution.
- The thesis and the work presented within it are the independent outcomes of my individual research efforts, and I have delineated the extent and scope of my contributions.
- This doctoral thesis is the result of my own independent research. Language aids such as Grammarly, QuillBot, and GPT 3.5 & 4 based large language models were utilized to enhance the clarity and coherence of the writing. Unless explicitly cited and referenced, the entirety of the content of methodologies, findings, and results, originates from my own research endeavors and experiments.
- I have duly acknowledged all main sources of assistance and financial support.

Signed: Felip Guimerà Cuevas

---

Date: February 28, 2024

---



# Artificial Intelligence for Automated Decision-Making in Error-Pattern Recognition

by Felip GUIMERÀ CUEVAS

## ABSTRACT

The demand for integrating **Artificial Intelligence (AI)** into diverse systems continues to expand rapidly. With growing reliance on AI, there is a constant need to deliver increasingly more dependable and robust solutions. This thesis aims to address common **Machine Learning (ML)** challenges and offers solutions in the field of automated decision-making and pattern recognition. It explores, presents, and analyzes novel methods and algorithms for **Representation Learning**, focusing on "Neural Architecture & Data Representation", "Manifold & Embedding Learning", and "Data Integration & Analysis".

The contributions of this thesis include<sup>1</sup>: (1) a novel neural architecture based on complex-valued neural networks, (2) a framework for encoding hierarchical one-to-many relationship databases into contextualized numeric data representations, (3) an adaptive and robust feature normalization and pre-processing technique, (4) a method for synthetic data augmentation on hierarchical databases, (5) a contrastive Representation Learning framework for tree structures, (6) extending and generalizing hierarchical Embedding Learning to multiple data views, (7) methods and diverse loss functions for Manifold Representation Learning and Clustering, (8) approaches for automated textual description generation for cluster groups, (9) a quality evaluation metric for clustering under coarse label uncertainty, (10) methods for determining representative textual labels for clustering accurate sensor data with inexact annotations, (11) a Transfer Learning approach to distilling insights from ML models trained on different data sets, (12) a novel approach to Out-of-Distribution Detection and Novelty Detection by leveraging mismatches in model calibration alignment, and (13) a method for representative sampling within quantile windows in data streams of unknown final length.

---

<sup>1</sup>A detailed list of contributions is given in 1.5 [Research Contribution Summary].



Center for Information and Language Processing  
LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

BMW GROUP

## DOCTORAL THESIS

In fulfillment of the requirements for the degree of Doctor of Philosophy.

### Artificial Intelligence for Automated Decision-Making in Error-Pattern Recognition

by Felip GUIMERÀ CUEVAS

#### CONTEXT & OVERVIEW

The rapid growth of available data and computational power has presented a huge opportunity for Artificial Intelligence (AI) to be integrated into various systems more effectively, robustly, and reliably than ever before. Although there are many well-established design patterns and promising solutions, there is still a strong need and desire to continuously develop more and more advanced AI models. Machine Learning (ML), a sub-branch of AI, employs algorithms and statistical models to learn and adapt based on available training data. ML is advancing rapidly, producing solutions that are increasingly stronger in many aspects: more successful, scalable, resilient, capable, efficient, etc.

This doctoral thesis is dedicated to addressing various challenges associated with Automated Decision-Making and Error Pattern Recognition within the field of ML and Deep Learning (DL). The contributions of this work build upon and go beyond typical or existing methods by extending, improving, and offering new alternative approaches. The research focuses on devising new (optimized) solutions for different relevant domains, particularly in Representation Learning. It takes into account diverse data types, including hierarchical graphs, images, data streams, and text; aiming to devise novel methodologies and approaches to provide new insights and generate optimized solutions.

A comprehensive literature review on many topics is provided, offering an inclusive introduction and a thorough understanding of the research's motivations and context. Within this context, the thesis introduces novel algorithms and refinements, highlighting issues within these domains and approaches, and proposes alternative solutions and improvements. The contributions offer valuable viewpoints and insights into general ML (i.e. DL), including manifold and embedding learning, latent space aggregations, neural architectures, outlier detection, clustering, graph-based learning models, and more.

**In summary:** this doctoral thesis aims to contribute to the advancement of AI methods (Chapter 1.5). It focuses on ML-based solutions for automated decision-making and error pattern recognition. The thesis introduces novel methods for manifold, embedding and representation learning, outlier detection, clustering, and graph-based DL. The goal is to help enhance AI model performance, efficiency, and robustness; across various data types and for common use cases.



Center for Information and Language Processing  
LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

BMW GROUP

# DISSERTATION

In Erfüllung der Anforderungen für den Abschluss des Grades Doctor of Philosophy.

## Künstliche Intelligenz zur automatisierten Entscheidungsfindung und Fehlermustererkennung

vorgelegt von Felip GUIMERÀ CUEVAS

### KONTEXT & ÜBERBLICK

Die stetig steigende Nachfrage nach der Integration von künstlicher Intelligenz (KI) in verschiedene Systeme reflektiert den starken Bedarf an automatisierbaren, effektiven, robusten und vertrauenswürdigen Lösungen. Mit dem kontinuierlich wachsenden Fortschritt des maschinellen Lernens (ML) und Deep Learnings (DL) werden immer bessere und leistungsfähigere Algorithmen und Methoden entwickelt, die auf einer Vielzahl von sehr unterschiedlichen Datentypen anwendbar sind. Dabei entstehen fortlaufend neue Möglichkeiten und Verbesserungen für immer robustere Lösungen, sodass der Wunsch und die Erwartung an noch leistungsstärkere und effizientere KI-Modelle stetig wächst, insbesondere um immer komplexere und anspruchsvollere Probleme lösen zu können. Dies wird vor allem durch gleichzeitige Fortschritte in der Hardware ermöglicht, welche die Verarbeitung größerer und umfangreicherer Datensätze und Datenmengen zunehmend effizienter macht, sodass immer größere und leistungsfähigere KI-Modelle angewendet werden können.

Diese Dissertation konzentriert sich auf die Bewältigung verschiedener relevanter Herausforderungen im Bereich der automatisierten Entscheidungsfindung und Fehlermustererkennung. Besonderer Fokus wird auf Representation Learning gelegt, um zur Optimierung verschiedener relevanter Bereiche beizutragen. Dabei werden eine Vielzahl von Datentypen wie hierarchische Graphen, Bilder, Datenströme und Texte berücksichtigt. Das Ziel ist es, durch neuartige oder verbesserte Methoden und Ansätze neue Erkenntnisse zur Generierung optimierter Lösungen beizutragen. Diese erlauben wertvolle Perspektiven und Einblicke in ML und DL, insbesondere in den Bereichen Manifold und Representation Learning, Ausreißer- und Anomalieerkennung, Clustering, Graphen-basierte Lernmodelle und mehr. Zudem bietet die Dissertation eine umfassende Literaturübersicht, um ein gründliches Verständnis der Methoden und Beweggründe der Forschung zu erleichtern.

Zusammenfassend: diese Doktorarbeit<sup>2</sup> zielt darauf ab, zum Fortschritt von KI-Methoden und ML-Modellen beizutragen, und konzentriert sich dabei auf die Bereiche automatisierte Entscheidungsfindung und Fehlermustererkennung, die in der Industrie von großer Bedeutung sind. Es werden verschiedene Algorithmen, Optimierungen, Anpassungen und Lösungsansätze präsentiert, um die Qualität, Effizienz und Robustheit von KI-Lösungen für verschiedene Anwendungsfälle und Probleme zu verbessern.

---

<sup>2</sup>Die Dissertation ist in Englischer Sprache verfasst. Eine kurze übersetzte Gesamtzusammenfassung auf Deutsch ist im Anhang D dargeboten.



Center for Information and Language Processing  
LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

BMW GROUP

## TESI DOCTORAL

En compliment dels requisits per a l'obtenció del grau de Doctor of Philosophy.

### **Intel·ligència Artificial per a la presa de decisions automatitzada i el reconeixement de patrons d'errors**

presentada per Felip GUIMERÀ CUEVAS

#### CONTEXT I RESUM

L'augment constant de la demanda per a la integració de la Intel·ligència Artificial (IA) en diversos sistemes reflecteix una necessitat crucial de solucions automatitzables, eficients, robustes i fiables. Amb el progrés continu en l'Aprenentatge Automàtic (AA) i l'Aprenentatge Profund (AP), s'estan desenvolupant algoritmes i mètodes cada vegada més sofisticats i potents, aplicables a una àmplia gamma de tipus de dades molt diversos. Això porta a noves possibilitats i millors emergents contínuament per a solucions cada vegada més efectives, alimentant el desig i l'expectativa de models d'IA encara més competents i optimitzats, especialment per abordar problemes cada cop més complexos i exigents. Aquest avanç és impulsat principalment per millors simultànies en el hardware, que fan que el processament de conjunts de dades més grans i extensos sigui cada vegada més eficient, permetent així l'aplicació de models d'IA (molt) més potents.

Aquesta tesi doctoral es centra en abordar diversos reptes rellevants en el camp de la presa de decisions automatitzada i el reconeixement de patrons d'errors. Es posa especial èmfasi en l'aprenentatge de varietats i la representació per contribuir a solucions optimitzades en diferents àrees rellevants. La tesi considera una diversa varietat de tipus de dades, com gràfics jeràrquics, imatges, fluxos de dades i textos. L'objectiu aquí és aportar nous coneixements a la generació de diverses solucions optimitzades mitjançant mètodes i enfocaments nous o millorats. La tesi ofereix perspectives i coneixements valuosos sobre AA i AP, especialment en àrees com l'aprenentatge de la representació de varietats, les agregacions de varietats, la detecció d'anomalies, la clusterització, els models d'aprenentatge basats en gràfics, etc. Finalment, la tesi també proporciona una revisió exhaustiva de la literatura per facilitar una comprensió profunda dels mètodes i de les motivacions de la recerca.

En resum, aquesta tesi doctoral té com a objectiu contribuir a l'avanç de mètodes i models d'IA, centrant-se en el camp de la presa de decisions automatitzada i el reconeixement de patrons d'errors en entorns industrials. Es presenten nous algoritmes, adaptacions i solucions amb l'objectiu de millorar la qualitat, l'eficiència i la robustesa dels models d'IA per a diverses aplicacions.



# Acknowledgements

I would like to state my sincere gratitude to the following individuals and organizations whose support was invaluable throughout my academic journey:

## Academic Mentors

- **Dr. Helmut Schmid:** I am deeply grateful for his consistent guidance, great feedback, and active engagement in technical discussions that positively influenced my doctoral thesis.
- **Prof. Benjamin Roth and Dr. Thomy Phan:** Their exceptional teaching during my graduate studies sparked my passion for Artificial Intelligence, especially in Machine and Deep Learning.

## Professional Support

- **Dr. Iryna Bastian and Dr. Raphael Weingartner:** I am thankful for the exceptional opportunity and trust they placed in me, providing constant personal support and encouragement throughout my doctoral research.
- **BMW Group:** Their confidence in me, provision of resources, and financial support have been instrumental in conducting my research. I am grateful for their significant contribution to my academic endeavors.

## Collaborators and Supporters

I extend my gratitude to all individuals with whom I collaborated, received feedback from, and engaged in technical discussions. Their insights and feedback contributions have greatly enriched my research and learning experience. Again, this holds particularly true for the BMW Group, as they fully financed this research project, and provided access to essential resources and data.

## Family Support

Finally, I express my heartfelt thanks to my **parents and family**. Their unwavering motivation, emotional support, and encouragement have been pivotal not just during my thesis but throughout my life. Their influence has played a defining role in shaping the person I am today, and for that, I am immensely grateful.

*“If a machine is expected to be infallible, it cannot also be intelligent.”*

— Alan Turing.

*“As our circle of knowledge expands, so does the circumference of darkness surrounding it.”*

- Albert Einstein

*“I refuse to say anything beyond five years because I don’t think we can see much beyond five years.”*

- Geoffrey Hinton



# Contents

<b>Declaration of Authorship</b>	iii
<b>Abstract</b>	v
<b>Context &amp; Overview (English)</b>	vii
<b>Kontext &amp; Überblick (Deutsch)</b>	ix
<b>Context i Resum (Català)</b>	xi
<b>Acknowledgements</b>	xiii
<b>Preface</b>	xxix
<b>1 Contributions</b>	1
1.1 Research Motivation . . . . .	1
1.2 Research Scope . . . . .	2
1.3 Research Challenges . . . . .	2
1.4 Research Contribution Scope . . . . .	2
1.5 Research Contribution Summary . . . . .	3
<b>2 Background</b>	7
2.1 Big Data . . . . .	7
2.2 Artificial Intelligence . . . . .	8
2.2.1 Definition & Philosophy . . . . .	8
2.2.2 Rule-based Systems . . . . .	9
2.2.3 Paradigms of Machine learning . . . . .	9
Unsupervised Learning . . . . .	10
Supervised Learning . . . . .	10
Self-Supervised Learning . . . . .	10
Reinforcement Learning . . . . .	11
Hybrid Learning Methods . . . . .	11
2.2.4 Weakly-Supervised Learning . . . . .	11
2.2.5 Challenges in Unsupervised Clustering Methods . . . . .	12
2.2.6 Rule-based vs. Machine Learning . . . . .	12
2.2.7 Model Generalization . . . . .	15
2.3 Data Encoding & Dimensionality . . . . .	15
2.4 The Curse & Blessing of Dimensionality . . . . .	18
2.4.1 The Curse of Dimensionality . . . . .	18
Distance Concentration . . . . .	19
Data Sparsity . . . . .	19
Combinatorics . . . . .	20
Other . . . . .	20
2.4.2 The Blessing of Dimensionality . . . . .	20

2.5	Data Augmentation . . . . .	21
2.5.1	Assessing Generated Data Quality . . . . .	21
2.6	Optimization . . . . .	22
2.6.1	Gradient Descent . . . . .	23
2.6.2	Back-Propagation . . . . .	25
2.7	Neural Networks . . . . .	26
2.8	Transformers . . . . .	28
2.9	Clustering & Kleinberg's Theorem . . . . .	31
2.10	Model Evaluation . . . . .	32
2.10.1	Generalization & Overfitting . . . . .	33
2.10.2	Clustering Evaluation . . . . .	33
	Intrinsic Clustering Evaluation . . . . .	33
	Extrinsic Clustering Evaluation & Uncertainty . . . . .	34
<b>3</b>	<b>Motivation</b>	<b>37</b>
3.1	Neural Networks Based on Complex Numbers with Weights Constrained along the Unit Circle . . . . .	37
3.2	Deep Learning for Hierarchical Databases with Recursive One-to-Many Relations . . . . .	38
3.3	Addressing Uncertainty and Completeness in the $B^3$ Cluster Evaluation Metric . . . . .	38
3.4	Automated Textual Description Generation of Clusters . . . . .	39
3.5	Normalization of Heterogeneous Feature Distributions . . . . .	40
3.6	Out-of-Distribution Detection . . . . .	41
3.7	Manifold Data Representation and Clustering . . . . .	42
3.8	Representative Data Stream Sampling for Trajectories . . . . .	42
<b>4</b>	<b>Introduction</b>	<b>45</b>
4.1	Framing the Context: Use Cases, Challenges, & Data . . . . .	45
4.1.1	Framing the Context: Example Use Cases . . . . .	45
	Plant Growth Tracking in Greenhouses: Multifaceted Observation Records . . . . .	45
	Hierarchical Data Structures: Entities, Relationships, and Attributes . . . . .	47
4.1.2	Framing the Context: Formal Definition . . . . .	48
4.1.3	Composition of Multiple Hierarchical Data Sources . . . . .	49
4.1.4	Data Quality . . . . .	50
4.1.5	Data Processing . . . . .	50
4.1.6	Problem Statement . . . . .	51
4.2	Previous Methods Employed in Real-World Applications . . . . .	52
4.2.1	Concerns . . . . .	52
4.3	Machine Learning for Automated Error-Pattern Detection . . . . .	53
4.3.1	Challenges in Machine Learning . . . . .	54
	Expensive Development . . . . .	54
	Data & Need for Custom Tailoring . . . . .	54
	Reasoning, Uncertainty, and Reliability . . . . .	55
4.4	Neural Networks Based on Complex Numbers with Weights Constrained along the Unit Circle . . . . .	56
4.5	Deep Learning for Hierarchical Databases with Recursive One-to-Many Relations . . . . .	57

4.6	Determining Representative Textual Labels for Clustering Accurate Sensor Data with Inexact Annotations . . . . .	59
4.7	Explainable and Interpretable AI . . . . .	61
4.7.1	Supervised Cluster Evaluation Metric . . . . .	61
4.8	Automated Textual Description Generation of Clusters . . . . .	64
4.9	Non-linear Normalization of Heterogeneous Feature Distributions with Adaptive Tanh-Estimators . . . . .	66
4.10	Transferring Knowledge Across Input Domains: Distilling Insights from Machine Learning Models Trained on Different Datasets . . . . .	69
4.11	Model Calibration for Out-of-Distribution Detection . . . . .	70
4.12	Manifold Data Representation and Clustering . . . . .	74
4.13	Representative Sampling in Data Streams . . . . .	76
<b>5</b>	<b>Preceding Project Endeavors</b>	<b>79</b>
5.1	Overview . . . . .	79
5.2	Data Encoding . . . . .	80
5.2.1	Categorical Univalent Features . . . . .	80
5.2.2	Tabular Data . . . . .	81
5.2.3	Text Data . . . . .	82
5.3	Dimensionality Reduction . . . . .	82
5.4	Error-Pattern Inference . . . . .	83
<b>6</b>	<b>Related Work</b>	<b>85</b>
6.1	Neural Networks Based on Complex Numbers . . . . .	85
6.2	Heterogeneous Graph Learning . . . . .	86
6.3	Robust Feature Normalization . . . . .	88
6.4	Model Output Calibration . . . . .	90
6.5	Supervised Clustering Evaluation . . . . .	93
6.6	Manifold Clustering . . . . .	94
6.7	Evaluating Generated Content against References . . . . .	95
6.7.1	BERT-score . . . . .	96
6.7.2	BART-score . . . . .	96
6.7.3	BLEURT-score . . . . .	97
6.8	Text Generation Strategies . . . . .	97
6.9	Large Language Models: Prompt Engineering . . . . .	98
6.10	Representative Sampling in Data Streams . . . . .	100
6.10.1	Background on Reservoir-Sampling . . . . .	100
	Reservoir-Sampling . . . . .	100
	Biased Reservoir-Sampling . . . . .	101
	Decaying Reservoir Sampling . . . . .	101
	Sliding Window Sampling . . . . .	101
	Quantiles in Data Streams . . . . .	101
	Spatio-Temporal Trajectories . . . . .	102
6.10.2	Related Work on Data Stream Sampling . . . . .	102
<b>7</b>	<b>Methods</b>	<b>105</b>
7.1	Neural Networks Based on Complex Numbers with Weights Constrained along the Unit Circle . . . . .	105
7.1.1	Convolutional Layer . . . . .	105
7.1.2	Numerical Stability . . . . .	105
7.1.3	Weight Initialization . . . . .	105

7.1.4	Alternative Normalization Correction Factor . . . . .	105
7.1.5	Enhanced Complex Neurons with Linear Weight Scaling . . . . .	105
7.2	Data Representation & Encoding . . . . .	106
7.2.1	Hierarchical Data Interpretation . . . . .	106
7.2.2	Tree-Data Encoding . . . . .	108
7.2.3	Non-Hierarchical Entity Encoding for Feature Vectors . . . . .	110
7.2.4	Composite Graphs: Shared Data Structures for Reducing Memory Overhead & Ensuring Uniqueness . . . . .	112
7.2.5	Encoding Determinism . . . . .	113
7.3	Feature Pre-Processing . . . . .	113
7.3.1	Numeric Feature Normalization . . . . .	114
7.3.2	Ideal Spread Factor . . . . .	116
	Trainable Spread Factor . . . . .	116
	Spread Value . . . . .	116
7.3.3	Time-Stamps Encoding . . . . .	117
7.3.4	Ordinal Feature Encoding . . . . .	117
7.4	Tree-Data Augmentation . . . . .	118
7.5	Self-Supervised Contrastive Representation Learning on Tree Structures	120
7.5.1	Loss Function . . . . .	122
	Reconstruction- & Composition Loss: Auto-Encoders . . . . .	123
	Similarity- & Contrast Loss: Embeddings . . . . .	124
	Classification Loss . . . . .	126
	Total Loss . . . . .	127
7.5.2	Neural Network Forward Call . . . . .	129
7.5.3	Embedding Layer . . . . .	130
7.5.4	Extraction Layer . . . . .	130
7.5.5	Batching . . . . .	131
7.5.6	Model Architecture . . . . .	131
7.5.7	Example . . . . .	132
7.5.8	Model Improvements . . . . .	133
	Adaptive Look-up Embeddings . . . . .	134
	Residual Attention Mechanism . . . . .	135
	Inception Networks . . . . .	136
	Siamese Networks . . . . .	137
	Mean Teacher Networks . . . . .	139
	Complex-Valued Neural Networks . . . . .	140
	Label Smoothing . . . . .	140
	Focal Smoothing . . . . .	142
	Combining Focal- & Label smoothing . . . . .	143
	Dropout . . . . .	144
	Class Imbalance . . . . .	144
	Feature Independence . . . . .	146
	Lipschitz Continuity & Spectral Normalization . . . . .	146
	Kernel Embeddings & Pooling Layer . . . . .	147
7.5.9	Efficient Contrastive Representation Learning . . . . .	151
	Negative Sampling-Based Contrastiveness . . . . .	151
	Class-Label Based Contrastive Representation Learning . . . . .	152
	Mean-Teacher Average Class-Label Contrastiveness . . . . .	153
7.5.10	Residual Information Preservation . . . . .	154
	Fully Connected Residual Blocks and Basis Learning . . . . .	154
7.5.11	Training & Inference . . . . .	157

7.5.12	Full Model Composition . . . . .	158
7.6	Concatenated Representation Learning . . . . .	160
7.6.1	Handling Duplicate Instances . . . . .	162
7.6.2	Memory Efficient Similarity Search . . . . .	163
7.6.3	Optimized Memory-Efficient Graph Batching . . . . .	164
7.7	Manifold Data Representation and Clustering . . . . .	166
7.7.1	Taylor Approximation Smoothing . . . . .	166
	Manifold Clustering . . . . .	172
7.8	Clustering Strategy . . . . .	172
7.9	Supervised Clustering Quality Evaluation . . . . .	173
7.9.1	Background . . . . .	174
7.9.2	$\alpha$ Max-B-Cubed: A Supervised Metric for Addressing Uncertainty in Cluster Evaluation. . . . .	174
7.9.3	Extension to Imbalanced Data Sets . . . . .	178
7.10	Automated Textual Description Generation of Clusters . . . . .	179
7.10.1	Formal Specification . . . . .	179
7.10.2	Contrastive Language-Image-Based Pooling . . . . .	180
	CLIP Gradient-Guided Pooling Search . . . . .	180
	CLIP Weighted Pooling . . . . .	181
7.10.3	Synthetic Boosting of Target Captions . . . . .	181
7.10.4	Caption Selection . . . . .	182
7.10.5	Quality Evaluation . . . . .	182
7.11	Prompt Manipulation on Large Language Models for Customized Text Summarization and Structured Analysis . . . . .	183
7.12	Determining Representative Textual Labels for Clustering Accurate Sensor Data with Inexact Annotations . . . . .	184
7.12.1	Harmonic Mean of Intuitive Heuristics . . . . .	184
	Cluster Centrality Measures for HDBSCAN . . . . .	186
7.12.2	Lowest Auto-Regressive Modelling Loss over Cluster Embedding . . . . .	187
	Example of a Loss Metric Explanation . . . . .	188
7.13	Transferring Knowledge across Input Domains: Distilling Insights from Machine Learning Models Trained on Different Datasets . . . . .	188
7.14	Model Output Calibration & Novelty Detection . . . . .	190
7.14.1	Score Adjustment . . . . .	193
7.14.2	Calibration Discrepancy with Label Aggregations . . . . .	193
7.14.3	Hybrid ODD: Embedding- and Prediction-based . . . . .	194
7.15	Percentile Ranked Scores . . . . .	195
7.16	Representative Data Stream Sampling for Trajectories . . . . .	195
7.16.1	Decaying Quantile Chain-Sampling . . . . .	196
7.16.2	Full Quantile-Sampling . . . . .	204
<b>8</b>	<b>Experimental Methods</b>	<b>207</b>
8.1	Hardware . . . . .	207
8.2	High Dimensional Data Visualization . . . . .	207
8.3	Adaptive Complex-Valued Bi-Nonlinear Neural Networks . . . . .	209
8.4	Adaptive Tanh-Normalization . . . . .	209
8.5	Determining Representative Textual Labels for Clustering Accurate Sensor Data with Inexact Annotations . . . . .	209
8.5.1	Quality Assessment of Textual Generations . . . . .	210
8.6	Model Output Calibration & Novelty Detection . . . . .	211

<b>9 Analysis &amp; Results</b>	<b>213</b>
9.1 Adaptive Complex-Valued Bi-Nonlinear Neural Networks . . . . .	213
9.1.1 XOR-Problem . . . . .	213
9.1.2 Minimal Networks & Expressive Power . . . . .	213
9.1.3 Classification . . . . .	213
9.2 Loss-Weighting . . . . .	213
9.3 Embeddings . . . . .	215
9.3.1 Clustering Representations . . . . .	215
9.3.2 Hierarchical Dependent Embedding Representations . . . . .	216
Large Errors at Root-Nodes (Upper Levels) . . . . .	217
Large Errors at Leaf-Nodes (Lower Levels) . . . . .	218
Implications . . . . .	218
9.4 Deep Learning for Hierarchical Databases with Recursive One-to-Many Relations . . . . .	218
9.4.1 Unsupervised Deep Clustering . . . . .	218
9.4.2 Supervised Classification . . . . .	219
9.4.3 Aggregation Function . . . . .	220
9.5 Manifold Clustering & Embedding . . . . .	222
9.5.1 Preservation of Identifying Structures . . . . .	223
Taylor Approximation Error During Training . . . . .	226
9.5.2 Taylor Approximated Clustering . . . . .	228
9.5.3 Non-Smoothness of DNN Landscapes and Taylor Approximation . . . . .	229
9.6 Supervised $\alpha$ Max- $B^3$ Clustering Evaluation Metric . . . . .	230
9.6.1 Comparison of $\alpha$ Max- $B^3$ and Traditional $B^3$ on Synthetic Data .	230
9.6.2 Imbalanced Data Set . . . . .	230
9.6.3 Evaluation of Automatic Uncertainty Determination . . . . .	231
9.6.4 Uncertainty Estimation and Extrapolation . . . . .	232
9.6.5 Results on Real-World Data . . . . .	233
9.7 Cluster Explainability using Natural Language . . . . .	235
9.7.1 Visual Illustration of Embedding Aggregations . . . . .	235
9.7.2 Generative Captioning Results of Aggregated Embedding Vectors . . . . .	235
9.7.3 Exploring Manifold Aggregation Strategies for Generating Representative Image Cluster Descriptions . . . . .	236
9.8 Normalization of Heterogeneous Feature distributions with Adaptive Tanh-Estimators . . . . .	240
9.8.1 Classification With Neural Networks . . . . .	240
9.8.2 Analytical Optimization . . . . .	242
9.9 Model Output Calibration & Novelty Detection . . . . .	245
9.9.1 Score Adjustment . . . . .	245
9.9.2 Results on Model Output Calibration & Novelty Detection .	246
9.10 Clustering Strategy: Prediction-based Sub-Clustering . . . . .	250
<b>10 Discussion</b>	<b>253</b>
10.1 Neural Expressive Power & Bi-Nonlinear Complex-valued Based Neural Networks . . . . .	253
10.2 Hierarchical Heterogeneous Graph Neural Networks for Deep Learning on Recursive One-to-Many Databases . . . . .	254
10.3 Leveraging Knowledge Distillation and Domain Adaptation in Training for Transfer Learning . . . . .	256

10.4	Robust Non-linear Normalization of Heterogeneous Feature Distributions with Adaptive Tanh-Estimators . . . . .	257
10.5	Supervised $\alpha$ Max- $B^3$ Clustering Evaluation Metric . . . . .	259
10.5.1	Determining Representative Textual Labels for Clustering Accurate Sensor Data with Inexact Annotations . . . . .	260
10.6	Automated Textual Description Generation of Clusters . . . . .	262
10.7	Misaligned Output Calibration for Out-of-Distribution Detection . . . . .	263
10.8	Representative Sampling in Data Streams . . . . .	265
10.9	Clustering on Latent Manifold Structures . . . . .	266
<b>11</b>	<b>Summary &amp; Conclusions</b>	<b>269</b>
11.1	Neural Architecture and Data Representation . . . . .	270
11.1.1	Complex-valued based Neural Networks . . . . .	270
11.1.2	Hierarchical Heterogeneous Graph Neural Networks . . . . .	270
11.1.3	Synthetic Data Augmentation for Tree Structures . . . . .	271
11.1.4	Contrastive Representation Learning & Loss-Weighting . . . . .	271
11.1.5	Concatenated Representation Learning & Separate Data Training	271
11.2	Feature Pre-Processing & Representation Learning . . . . .	272
11.2.1	Robust Non-linear Normalization of Heterogeneous Feature Distributions with Adaptive Tanh-Estimators . . . . .	272
11.2.2	Completeness and Uncertainty in Cluster Evaluation . . . . .	272
11.2.3	Representation Learning and Manifold Clustering . . . . .	272
11.2.4	Manifold Aggregations of Latent Spaces for Generating Textual Descriptions of Clusters . . . . .	273
11.2.5	Determining Representative Textual Labels for Clustering Accurate Sensor Data with Inexact Annotations . . . . .	273
11.3	Knowledge Transfer, Outlier Detection, and Sampling . . . . .	274
11.3.1	Transfer Learning via Knowledge Distillation . . . . .	274
11.3.2	Misaligned Calibration for Out-of-Distribution Detection . . . . .	274
11.3.3	Representative Sampling in Data Streams . . . . .	274
11.4	Concluding Words . . . . .	275
<b>A</b>	<b>Out-of-Distribution Detection</b>	<b>277</b>
A.1	Proofs . . . . .	277
<b>B</b>	<b>Clustering</b>	<b>279</b>
B.1	Unsupervised Clustering of Hierarchical Databases . . . . .	279
B.2	Proofs . . . . .	280
B.2.1	Theorem 1 . . . . .	280
B.2.2	Proposition 1 . . . . .	280
B.2.3	Corollary 1 . . . . .	281
B.2.4	Theorem 2 . . . . .	281
B.2.5	Proposition 2 . . . . .	281
B.2.6	Corollary 2 . . . . .	282
B.2.7	Generalization to Multiple Clusters . . . . .	282
B.2.8	Proof of Coherence for $\alpha$ Max- $B_\delta^3$ . . . . .	283
B.2.9	Proof of Non-Monotonicity . . . . .	283
B.2.10	Theorems' Inequality Implications . . . . .	284
B.2.11	Proof of $\alpha$ -Plateau . . . . .	285

<b>C From Missteps to Milestones: Understanding Failed Attempts</b>	<b>287</b>
C.1 Failed Prototype Attempt: Automated Textual Description Generation	287
C.1.1 Methods . . . . .	287
Training objective . . . . .	288
Cluster-wise gradient accumulation . . . . .	289
Prototype discussion . . . . .	289
C.2 Sensor-Driven Textual Observation Report Prediction . . . . .	290
C.2.1 Prototype Discussion . . . . .	290
<b>D Deutsche Zusammenfassung</b>	<b>293</b>
D.1 Zusammenfassung . . . . .	293
D.1.1 Forschungsmotivation . . . . .	293
D.1.2 Forschungsbereich . . . . .	294
D.1.3 Forschungsherausforderungen . . . . .	294
D.1.4 Beitragsumfang der Arbeit . . . . .	295
D.1.5 Zusammenfassung der Beiträge . . . . .	295
NEURONALE ARCHITEKTUR UND DATENREPRÄSENTATION . . . . .	295
DIMENSIONSREDUKTION UND DARSTELLUNGSSUCHE . . . . .	296
DATENINTEGRATION UND ANALYSE . . . . .	297
<b>Bibliography</b>	<b>299</b>

# List of Figures

2.1	Example of a rule-based decision tree. . . . .	9
2.2	Supervision problematic . . . . .	12
2.3	AI paradigms . . . . .	14
2.4	Space partition . . . . .	20
2.5	Convex vs. Non-Convex . . . . .	23
2.6	Gradient descent . . . . .	24
2.7	Neural network . . . . .	26
2.8	Stacked Neural network . . . . .	27
2.9	Transformer model . . . . .	29
3.1	Using natural language to explain clusters. . . . .	40
4.1	Data dependency relations. . . . .	46
4.2	Illustrating the data acquisition process. . . . .	47
4.3	Hierarchical data structure composed of a primary root entity containing subsidiary elements. . . . .	48
4.4	Abstract data dependency relations. . . . .	48
4.5	Different HDBs integrated into a larger unified DB. . . . .	50
4.6	Big picture of applying ML. . . . .	54
4.7	Quality implications based on deterministic and non-deterministic cluster aggregations and separations. . . . .	64
4.8	Second-order Taylor approximation error (TA-error) from a source to other points in a simple 2D space. Points with larger TA-errors, or that are distant from the source, are less likely to <i>align</i> with the source. This can be used as a pairwise measure. . . . .	76
4.9	A manifold clustering illustration, where nearby points on a shared (smooth) manifold yield pairwise good (aligned) Taylor approximations. Proximity and locally good approximations indicate clusters. Proximity in Euclidean space differs from proximity in the manifold space. Same-colored points share the same local manifold. . . . .	77
5.1	Overview of the pipeline proposed in (agreement, n.d.). . . . .	80
5.2	Cardinality linearization from (agreement, n.d.) . . . . .	81
7.1	Graph interpretation of data. . . . .	106
7.2	Node features. . . . .	107
7.3	Data base database graph. . . . .	108
7.4	Node encoding. . . . .	110
7.5	Non-hierarchical entity modification. . . . .	111
7.6	Composite graphs. . . . .	112
7.8	Graph-data augmentation. . . . .	118
7.9	Non-transitiveness of the Jaccard metric. . . . .	121
7.10	Reconstruction loss using Auto-Encoders. . . . .	123

7.11	Classification loss. . . . .	127
7.13	Recursive model architecture. . . . .	132
7.14	Full recursive model architecture. . . . .	133
7.15	Example of embedding generation. . . . .	134
7.16	Adaptive look-up embeddings. . . . .	135
7.17	Attention integration. . . . .	137
7.18	Output-Inception. . . . .	138
7.19	Siamese networks and self-attention. . . . .	138
7.20	Attention integration. . . . .	154
7.21	Information preservation . . . . .	155
7.22	Information preservation. . . . .	155
7.23	Virtual nodes . . . . .	156
7.24	Full custom model composition. . . . .	158
7.25	Embedding concatenation and model forwarding. . . . .	160
7.26	Suggested approach for separating data into train and test sets after identifying and removing duplicates to ensure data split consistency across different data views. . . . .	162
7.27	Similarity search using different methods. . . . .	164
7.28	Using implicit nested matrix representations to avoid unnecessary and excessive padding in batch processing. . . . .	165
7.29	Illustration of the manifold approximation concept. . . . .	168
7.30	Illustration of the main architecture of ATLAS. . . . .	171
7.31	Performing sub-clustering on clusters to extract deeper and finer sub-patterns. . . . .	173
7.32	Conceptual illustration of the cluster aggregation and consolidation step for generating super-sets. . . . .	175
7.33	A graphical representation demonstrating the behavior of $\eta_{\alpha}^{[j]}$ across varying values of $\alpha$ using a fixed configuration of points as an example. . . . .	177
7.34	The full training architecture for generating CLIP-search-based cluster descriptions. . . . .	181
7.35	The Harmonic Clustering Score as the harmonic mean of three intuitive score heuristics. Ranking by this score identifies the most "representative" elements; since achieving a good score requires all three heuristics to yield high scores. . . . .	186
7.36	Identifying the optimal representative pair within a cluster by sorting pairs according to their auto-regressive modeling loss relative to their cluster's embedding . . . . .	187
7.37	Transferring knowledge across input domains by distilling insights from models trained/fitted on different data sets. . . . .	189
7.38	Sequence chaining for $\phi = 1$ . . . . .	197
7.39	The single-path forward chaining probability for various quantile values $\phi$ . . . . .	199
7.40	Inverse weighting relative to reachability likelihood. . . . .	200
8.1	An example of how multiple visualizations using UMAP on the same data may provide different outcomes. . . . .	208
9.1	Comparing the distribution of adjusted and multiplied weights derived from random losses. . . . .	215
9.2	Comparing the distribution of adjusted and multiplied weights derived from random losses. . . . .	215

9.3	Projecting clustered random vectors. . . . .	216
9.4	The embedding projection and the closest points of random instances within some clusters are visualized. The images of the instances are from the data set (Rebrickable®, 2022). . . . .	219
9.5	The results of a classification and semi-supervised task are displayed for the ten most frequent classes. The upper graphs relate to the Rebrickable® DB; the lower ones to the synthetic data. . . . .	220
9.6	Projection of latent representations for <i>FMNIST</i> using different architectures. Classification labels were obtained by splitting the original ten label indices into two classification groups A & B ( based on whether the label indices are even or odd). . . . .	224
9.7	An example of too-weak AEs that fail to effectively interpret the underlying data and preserve structural identity as intended (finding and preserving sub-labels). . . . .	225
9.8	Accuracy and normalized Taylor approximation error during training on three different architectures. . . . .	227
9.9	Comparison between HDBSCAN clustering on the Euclidean distance vs. AP using the TA error. . . . .	228
9.10	Multiple clustering assignments are evaluated using clustering scores and depicted visually. The optimal clustering consists of five classes. A finer and purer sub-clustering should have better scores compared to a coarser and less pure clustering. The proposed $\alpha$ Max- $B^3$ metric, being a variation of the standard $B^3$ , generates more robust and fair scores. It prioritizes correct extraction of sub-clusters over incorrect super-clusters, while still penalizing non-homogeneous super-clusters when the number of sub-clusters becomes excessive. . . . .	231
9.11	The $B^3$ , $\alpha$ Max- $B^3$ , and $\alpha$ Max- $B_\delta^3$ scores on an imbalanced data set of five labels. The performance was evaluated based on different cluster counts $k$ . On coarse clusters ( $k \leq 5$ ), $B^3$ and $\alpha$ Max- $B^3$ perform equally. However, $\alpha$ Max- $B^3$ performs fairer on finer sub-clusters ( $k \geq 5$ ) than $B^3$ since it gives more weight to sub-clusters under uncertainty. $\alpha$ Max- $B_\delta^3$ accounts for class imbalance; which results in differences in $k < 5$ . . . . .	232
9.12	A comparison of different pooling operations on UMAP projected image cluster groups from synthetic data and the COCO data set. Depicted are: mean-, median-, max-, GM-, and instance GM pooling. . . . .	236
9.13	[Extract 1] A sample of CCs generated from TextCaps data (Sidorov et al., 2020) using diverse pooling techniques; with corresponding BERT-scores. Cluster captions were produced by passing the pooled representation through a decoder, applying a sampling decoding strategy, and choosing the caption with the highest BERT-score. Four clusters are shown, each with a random subset of nine cluster instances. . . . .	237
9.14	[Extract 2] A sample of CCs generated from TextCaps data (Sidorov et al., 2020) using diverse pooling techniques; with corresponding BERT-scores. Cluster captions were produced by passing the pooled representation through a decoder, applying a sampling decoding strategy, and choosing the caption with the highest BERT-score. Four clusters are shown, each with a random subset of nine cluster instances. . . . .	238
9.15	Comparing normalized output distributions and performance. . . . .	241
9.16	Results on classification tasks. . . . .	241
9.17	Results on classification tasks. . . . .	242
9.18	tanh-normalization using different spread values. . . . .	243

9.19	Wasserstein loss against different spread values. . . . .	244
9.20	Class distances in an embedding space . . . . .	246
9.21	An embedding projection showing inliers and outliers. . . . .	249
9.22	Analyzing sub-cluster data hierarchies in Sklearn’s diabetes data set; identifying core points, clusters, and outliers. . . . .	250
9.23	Analyzing sub-cluster data hierarchies in Sklearn’s California housing data set; identifying core points, clusters, and outliers. . . . .	250
B.1	The embedding projection and the closest points of random instances within some clusters are visualized. The images of the instances are from the data set (Rebrickable®, 2022). The results are independent of the ones shown in Figure 9.4.1; it depicts a completely different training run (on the same data set). . . . .	279
C.1	An overview of an initial prototype training architecture pipeline (i.e. first attempt) for generating cluster descriptions. . . . .	288
C.2	Learning from textual observations and training decoder models for automatic text generation out of sensor data. . . . .	290

# List of Tables

2.1	Top Large Language Models . . . . .	30
9.1	Class-weighted F1 classification scores of various models with different model sizes and learning rates. $\mu$ represents the model size ratio to BERT-mini for a given model; $\eta$ is the learning rate. Bert was used without positional encoding; token order was randomly sampled each time before truncation. The median sequence length was 234 (mean $\sim$ 367). Bert employed the default maximum sequence length of 512. However, to emulate a situation where the textual DB encoding exceeds the sequence length limit, a modified version with a limit of 128 (a fourth of the maximal sequence windows) was also evaluated; denoted by *Bert. . . . .	221
9.2	A comparison of scores against ground truth uncertainty. $n$ denotes the number of real sub-clusters merged under the same visible coarse label. Automatically determining $\alpha$ yielded here the same scores as using the real uncertainty value (i.e. $\alpha = \frac{n-1}{n}$ ). . . . .	233
9.3	A comparison of scores from merged data sets of varying levels of uncertainty (sub-cluster tuples) under different degrees of noise, number of instances, and cluster classes. $(m, n)$ denotes $m$ coarse labels; each comprising $n$ sub-clusters. $\mathcal{O}\frac{n-1}{n}$ is the average uncertainty of both merged data sets. $c$ denotes the total number of total number of real classes. . . . .	233
9.4	Estimating data uncertainty by extrapolating $\alpha$ and comparing it to the actual uncertainty. The extrapolation was perfect for consistent data sets with a uniform cluster uncertainty. For inconsistent data uncertainty, e.g. merging two different sets with different uncertainties, the extracted $\alpha$ was not perfect but approximated the uncertainty well. . . . .	234
9.5	Comparison of model evaluation on real and coarse labels for CIFAR100 sub-labels using $B^3$ and $\alpha$ MaxB-CUBED; both, for automatic estimation of $\alpha$ and using the real $\alpha$ . $VIT_{b-32}$ (Dosovitskiy et al., 2020) was used as base model; at multiple different accuracy levels. . . . .	234
9.6	Average text similarity scores for different pooling methods using various metrics with different scales. Higher scores indicate better performance; the average and maximum cluster-wise score using a brute-force search is included as a reference. . . . .	239
9.7	The average relative rank (smaller is better) of the scores among all methods per decoding strategy. . . . .	240
9.8	Approximate probability density coverage. . . . .	244
9.9	Ideal spread values. . . . .	245
9.10	Average Out-of-Distribution Area Under the Receiver Operating Characteristic Curve (ROC AUC) scores on <i>trivial</i> synthetic clusters for multiple simple classifiers. . . . .	246



# Preface

**Disclaimer:** This doctoral thesis was made possible thanks to a joint collaboration between the Ludwigs Maximilians University Munich and the BMW Group as part of an Industrial Doctoral Program. Financial support was provided by the BMW Group throughout the entire collaboration period. Any possible privacy-related information was either omitted or anonymized in compliance with the BMW Group's guidelines. Large language models contributed to maintaining grammatical accuracy and fluency in the composition; as the author of the thesis, I (Felip GUIMERÀ CUEVAS) retain full content responsibility.

**Target Audience:** This thesis represents the culmination of years of specialization and focused dedication in the field of Artificial Intelligence (AI). It is an exciting moment for me to present this work to the academic community. Through my thesis, I have come to especially appreciate the open research community in the field of AI, which has enabled me to keep pace with the latest developments in this fast-changing field. Being involved in the forefront of AI research has been a very exciting experience. Hence, a key endeavor in the writing of this thesis was to make it also accessible to a diverse range of readers (with varying levels of familiarity with AI). It can be read and understood without any (major) background in the subject, from start to finish. The thesis is of particular interest to those interested in Representation Learning and extracting meaningful patterns from data.

**Terminology:** Philosophically speaking, this dissertation, authored by "me" (Felip GUIMERÀ CUEVAS), stands not only as a reflection of my own research endeavors but also acknowledges the invaluable support, guidance, and contributions of the academic community and my loved ones (emotional support). Therefore, the use of "We" is both justified and necessary to recognize the collective effort that has made this work possible; and will, thus, be employed throughout this thesis. For instance, instead of stating "I propose", the expression "We propose" will be utilized. This choice of terminology implicitly includes the influence of numerous academic articles that have shaped or inspired my research (Bibliography). In essence, this thesis adopts the use of "We" in place of "I / Me".



## Chapter 1

# Contributions

This doctoral thesis explores, discusses, and proposes various (new) methods and algorithms in the field of Artificial Intelligence (AI), which will be introduced in Chapter 7 [Methods]. It is essential for readers, especially those without a Deep Learning background, to have a basic understanding of the reasons behind their proposal and usage, as well as their interrelationships and contextual placement within AI. Therefore, it is important to first establish a foundation of fundamental concepts and terms in the AI field before going into deeper mathematical and technical details. The fast-paced nature of AI development makes it difficult to keep up with the latest advancements. For those entering the field, it is imperative to not only remain aware of the most recent publications and breakthroughs but also to discern where to commence their literature review and learning. Basic background knowledge in this immense field is particularly essential for situating this thesis and its AI methods within a *context*. The goal of this thesis is to pave the way for further advancements and solutions in the field of AI. An inclusive literature overview not only facilitates this process but also guarantees the effective use of pre-existing knowledge, thereby reducing redundancy in research efforts in future work.

To bridge the gaps in understanding, this thesis will provide different sections that will motivate the need for different AI methods and solutions, highlight *why* we need them, provide the necessary mathematical and algorithmic background, outline variants and previous work, and illustrate how AI solutions can be approached by diverse methods. Overall, the thesis presents related work of relevant literature, but the reader is strongly encouraged to (extensively) explore the cited references and literature reviews to gain an even deeper understanding of the topic, improve overall comprehension, and situate the thesis into the right *context*.

### 1.1 Research Motivation

This thesis is the result of a previous data analysis project that demonstrated the potential of AI methods in terms of efficiency and effectiveness (see Chapter 5). This led to the current thesis, which was first aimed at improving or completely redesigning the previous model and methodology to obtain even more robust and efficient results. Strong and scalable AI solutions are highly desirable due to their broad applicability and adaptability to various complex problems, such as prediction, classification, similarity search, semantic search, outlier detection, and out-of-distribution detection, particularly relevant to the industry. In this regard, deep embeddings play a crucial role and represent contextualized numerical representations of entities. Obtaining good numerical representations is challenging, but it allows for immense potential in many different applications and is, thus, incredibly impactful. Once such an AI model architecture or algorithmic method is developed for generating deep

embeddings, it can be applied to many different use cases; making research in AI methods a highly rewarding investment.

## 1.2 Research Scope

The research scope covered in this thesis includes various domains such as Machine Learning, Deep Learning, Embeddings, Manifold Learning, Outlier Detection, and Clustering. Nonetheless, the primary project branch is Representation Learning; which here includes identifying meaningful and efficient representations for complex unconventional data type instances and data structures. From this primary branch, diverse sub-branches have emerged, addressing, improving, and tackling different sub-topics and applications/downstream tasks; either directly or indirectly related to Deep Embeddings or Manifold Learning, yet always situated within the domain of Machine Learning.

## 1.3 Research Challenges

Evaluation is a fundamental aspect of scientific research since it allows us to assess the quality of a method and compare it to other methods. Any method without representative or formal means of quantitative assessment (or comparison to other methods) may be deemed useless. However, in the field of Machine Learning, evaluating methods is challenging due to the trade-offs between the inherent advantages and disadvantages of different methods in many aspects. Different methods typically excel in different tasks, and some may e.g. be more efficient but less effective; or the other way around. Thus, conducting a "fair" evaluation is not always straightforward, and biased evaluations can lead to incorrect conclusions. For example, in clustering, different algorithms may perform well on different data sets, and there is no universal algorithm that outperforms all others in every aspect.

Therefore, it is essential to not only rely on empirical results but also use mathematical verification or proofs. Mathematical proofs guarantee certain behaviors, unlike empirical results (which require extensive evaluation to confidently assess reliability and significance). Hence, in this thesis, we aimed to find a balance between empirical results and mathematical validation; i.e., when appropriate.

## 1.4 Research Contribution Scope

The research presented in this thesis constitutes an essential part of an industrial doctoral project program that combines theory with practical applications. However, this thesis specifically concentrates on encapsulating the theoretical and analytical groundwork carried out within the project; and does *not* provide privacy-sensitive information. It presents theoretical solutions and approaches that were developed to address the underlying research problem. The thesis itself is entirely centered on providing foundational theoretical knowledge and does, in particular, not include implementation or application details. This separation between theoretical contributions and implementation details allows for a clear presentation of foundational academic knowledge, which ensures that the concepts can be applied beyond the current project and be generalized to other (different) use cases.

The key contributions of this thesis are outlined in the methods and results chapters, along with their respective discussions. The motivation, introduction, and related work sections serve to provide a comprehensive overview of previously published studies in the academic literature, which entail common background knowledge that we deem fundamental for situating this thesis into context. While extensive references are provided for these related works and introductory chapters, many aspects are generally considered basic and already well-known background knowledge. To differentiate our work from previous studies, and highlight our contributions in this thesis, all our main (novel) contributions, starting primarily from the methods chapter, are summarized and delineated in the paragraphs below:

## 1.5 Research Contribution Summary

The following are the primary *novel* academic contributions of this thesis to the knowledge in the field. This is in addition to our literature review contribution but constitutes original "novel" methodologies. Briefly summarized, we have divided our work into different main areas:

- NEURAL ARCHITECTURE AND DATA REPRESENTATION
  - **Introduced a novel neural architecture based on complex-valued neural networks (Chapter 7.1):** Traditional neural networks use real-valued parameters to process and analyze input data. Our novel complex-valued neural architecture uses complex-valued-based equivalent parameters instead. This architecture offers several unique properties over traditional neural networks, including the very interesting property that neural inputs can never be surpassed or eliminated by weights.
  - **Proposed a framework for encoding hierarchical one-to-many relationship databases into contextualized numeric data representation (Chapter 7.2):** One-to-many relationship databases, common in many domains, can be challenging to encode effectively and preprocess for Machine Learning algorithms and applications. Our framework uses a hierarchical data representation approach that maps the one-to-many relationships into a unified tree structure. This approach enables us to process, analyze, and learn from one-to-many relationship databases efficiently and effectively.
  - **Proposed an adaptive and robust feature normalization and preprocessing method (Chapter 7.3):** Effective feature normalization and preprocessing methods are crucial for improving and ensuring the performance of Machine Learning models. We focused on enhancing robustness and training convergence by reducing the effect of outliers and improving feature scaling (FS). Specifically, we achieved this by introducing an improvement to the Tanh-Normalization method for feature values that calculates an *optimal* scaling, thereby avoiding the need for (unnecessary) adjustments to model weights due to an initially poor, sub-optimal FS.
  - **Developed a method for synthetic data augmentation on hierarchical databases (Chapter 7.4):** Data augmentation is a crucial technique used to improve the performance of Machine Learning algorithms. Our method generates synthetic data samples for tree structures for (complex) hierarchical databases. This enables us to increase the size and diversity of the training data to improve the performance of models trained on this data.

- MANIFOLD AND EMBEDDING LEARNING

- **Introduced a contrastive Representation Learning framework for tree structures (Chapter 7.5):** Representations are crucial for the success of many Machine Learning tasks. Our contrastive framework can be applied to (hierarchical) tree structures to learn more representative embeddings while being flexible in operating in an unsupervised, semi-supervised, supervised, or even self-supervised manner. This approach has the potential to extract meaningful embeddings for hierarchical database entities to enable the application of further downstream tasks that require contextualized numerical representations.
- **Extended and generalized the hierarchical database embedding learning framework to unify multiple data views through concatenated Representation Learning (Chapter 7.6):** Concatenated Representation Learning is a technique where embeddings learned from different parts of the input data are concatenated to form a unified intermediate representation, from which a final embedding is then learned or derived. Our solution thus is to concatenate multiple data views from different hierarchical databases into a single unified database, by first computing the database representations of the individual hierarchical data views and then learning a joint embedding representation for further downstream tasks. This also enhances the scalability and adaptability of our framework to larger and more intricate databases.
- **Designed methods and diverse loss functions for manifold Representation Learning and Clustering (Chapter 7.7):** Manifold Representation Learning and Clustering are particularly important for high-dimensional data with complex structures. Our methods use (i) manifold learning techniques to reduce dimensionality while preserving important information and, (ii) clustering algorithms to partition the data set into groups. This is done using multiple loss functions together in a special neural training architecture (particularly designed for hierarchical data).
- **Developed approaches for automated textual description generation of cluster groups (Chapter 7.10):** Cluster groups can be challenging to interpret, especially in high-dimensional data. Our approaches generate automated textual descriptions that provide useful insights into the clustering group's characteristics and help human users to better understand the clusters' properties. These descriptions provide a single, concise, and natural summary of each cluster, conveying the essence without delving into specific (different) details about individual cluster instances.

- DATA INTEGRATION AND ANALYSIS

- **Introduced a  $B^3$ -based supervised evaluation metric for clusters under label uncertainty (Chapter 7.9.2):** Clustering under label uncertainty (i.e. inexact, coarse labels) is a challenging problem that is commonly encountered in many real-world applications and data sets. Our proposed quality evaluation metric for clusterings is specifically designed to account for coarse labels. It draws attention to a potential flaw in the widely-used  $B^3$  cluster evaluation metric.  $B^3$  assumes and satisfies multiple desired, intuitive formal clustering constraints that, however, may not be optimal or accurate when dealing with poor, imperfect, and uncertain labels. This is

particularly relevant and the case for coarse labels, where multiple entities may share the same (super-) label under some degree of uncertainty.

- **Proposed methods for determining representative textual labels for clustering accurate sensor data with inexact labels (Chapter 7.12):** Accurate sensor data with inexact labels is a challenge that arises in many real-world applications. Our proposed methods determine representative textual labels that summarize the data's characteristics, even when the data set has inexact labels. For example, it is often of interest to rank textual annotations based on their suitability to represent identified clusters in high-quality sensor data. However, the textual data may be potentially different in terms of quality and, particularly, not aligned with the sensor data. The goal here may then be to find the best representative annotation while accounting for these uncertainties in the textual quality.
- **Proposed a transfer learning approach to distilling insights from Machine Learning models trained on different data sets (Chapter 7.13):** Transfer learning is a powerful method that enables us to use the knowledge gained from one data set to enhance the performance of Machine Learning models trained on other data sets. This becomes particularly useful when the original training data is unavailable, and/or retraining on a new data set would compromise the insights learned from the original. Our approach, based on knowledge distillation, distills the necessary insights from pre-trained models and effectively transfers them to new models. This approach also works well when different units have their unique data sets with similar or equal objectives/labels. By training individual models on their respective data sets, the insights learned can then be e.g. shared with a master model. This distillation is also useful e.g. when one model has been trained on an easier but less precise data set but one wants to learn from a more complex but more accurate and precise data set, while still leveraging and incorporating the knowledge extracted from the easier data set.
- **Introduced a novel approach to out-of-distribution detection and novelty detection, leveraging mismatches in model calibration alignment (Chapter 7.14):** Model calibration is typically essential for accurate predictions, but we argue that mismatches in calibration can also be exploited to detect out-of-distribution samples or novelties in samples. Our proposed approach uses precisely these mismatches to detect out-of-distribution samples and outliers with high accuracy in comparison to other methods.
- **Developed a method for representative sampling within quantile windows in data streams of unknown final length (Chapter 7.16):** Sampling representative data streams is challenging, particularly when the final length of the stream is unknown. Sampling from data streams of unknown sizes is considered to be non-trivial due to dynamic data characteristics, limited memory, time constraints, and potential for sampling bias. Specialized algorithms and techniques have been developed to address different challenges such as via reservoir sampling and count-based sampling. We go a step further and address the problem of quantile sampling, which involves sampling from a data stream within a sliding non-fixed quantile window. We demonstrate how our proposed quantile sampling algorithm aligns with specific mathematical sampling properties.



## Chapter 2

# Background

### 2.1 Big Data

The number of available, accessible, and collectible data is growing yearly at an exponential rate (Chen, Mao, and Liu, 2014) and typically entails massive amounts of unstructured data. From data, one can extract knowledge, and from knowledge one can obtain new insights and value. However, identifying, extracting, and obtaining such knowledge from raw data are not the only challenges that arise. Managing, processing, and handling given data sets effectively and efficiently are only some of the very many complex challenges that need to be addressed and overcome (McAfee et al., 2012; Marx, 2013; Wu et al., 2013). Among these challenges characterizing Big Data (Kapil, Agrawal, and Khan, 2016), the six most key and well-known ones are:

- **Value:** The utility and value of data might be hidden or unstructured and usually need to be extracted and identified.
- **Volume:** Data comes in huge volumes and must be stored and accessed effectively, requiring scalable solutions for handling size and processing capacity.
- **Velocity:** Data changes and arrives quickly, exhibits rapid changes and influxes, and needs to be processed efficiently.
- **Variety:** Data can have different forms, structures, encodings, frequencies, and taxonomies, requiring solutions handling its heterogeneity.
- **Veracity:** The quality of data is not always reliable and might be considered questionable, conflicting, noisy, or even false.
- **Variability:** The structure or meaning of data can often change rapidly, leading to inconsistencies or fluctuations that may occur over time.

These challenges are commonly referred to as the *Six-Vs* (Andreu-Perez et al., 2015). The buzzword "*Big Data*", therefore, relates overall to all gathering and accumulations of massive data that cope with such complex challenges and deal with the explosion of available information and data streams. The process of actually analyzing and extracting knowledge is often referred to as "*Data Mining*" to find patterns or summarize the data, such that it becomes understandable and useful. Typical patterns (i.e. relationships, models, etc.) of interest are often correlations, rules, clusters, graphs, (hierarchical) tree structures, and recurrent patterns (Hand, 2007).

Data itself plays a central role in *Artificial Intelligence* (AI), especially in the sub-field of *Machine Learning* (ML), which will be introduced and discussed in the next section. Particularly ML algorithms rely on extensive data sets to identify these patterns and make meaningful and correct predictions.

**Take-Away**

The growth of available data is exponential and presents challenges in terms of managing its **volume**, **velocity**, **variety**, **veracity**, and **variability**. These challenges, known as the **Six-Vs**, are key in the field of Big Data, which involves analyzing and extracting knowledge from large amounts of unstructured data. Data plays a central role in Artificial Intelligence, particularly in Machine Learning.

**Keywords**

Big Data • Data Analytics • Challenges • Six-Vs • Unstructured Data.

## 2.2 Artificial Intelligence

### 2.2.1 Definition & Philosophy

*Artificial intelligence* (AI), although nowadays being used permanently under many contexts and highly hyped thanks to influential marketing sources, as well as due to hitting more and more headlines after consistently achieving incredible results (Wang et al., 2016a; Campbell, Hoane Jr, and Hsu, 2002; Buchanan, 2005) on many completely distinct fields and domains in industry and research, the technical and philosophical interpretation (Sloman, 1971; Haugeland, 1997) and definition of AI itself remains a debatable rabbit-hole with many rightfully distinct interpretations and, therefore, varies widely from literature to literature. Over the past few decades, various approaches and efforts have emerged to identify, define, characterize, and implement new and improved AI systems (Minsky, 1961; McCarthy, 1998; Russell and Norvig, 2002; Nilsson, 2014).

To convey the essence of AI to different audiences, I'd use varying levels of complexity and detail: If I was forced to define AI in three words to make a **child** understand AI, I would cite my personal, subjective favorite, interpretation from the MIT OpenCourseWare 6.034 (Winston., 2010):

**THINKING, PERCEPTION, REACTION**

If I was forced to define AI in a sentence to a **grown up** understand AI, I would then cite the more advanced definition from Winston., 2010:

**REPRESENTATIONS THAT SUPPORT MODELS TARGETED AT:**  
**THINKING, PERCEPTION, REACTION**

And I was ultimately forced to define AI for a Computer Science **student**, I would cite the full definition from Winston., 2010:

**ALGORITHMS ENABLED BY CONSTRAINTS EXPOSED BY**  
**REPRESENTATIONS THAT SUPPORT MODELS TARGETED AT:**  
**THINKING, PERCEPTION, REACTION**

Yet, if I was forced to answer "How does AI work?" in one word, I'd opt for:

**MATHEMATICS**

Interestingly, if you asked an AI large language model like GPT 3.5 Turbo (Abdullah, Madain, and Jararweh, 2022) to define AI in three words, it outputs e.g. "Learning, predicting, adapting"; and as a sentence: "AI processes data using algorithms to learn patterns, enabling it to make predictions or decisions without explicit programming".

In technical terms, AI can be divided into two main approaches: **Rule-based Systems** (RBS) and **Machine Learning** (ML). Although both come under the umbrella of AI, they are fundamentally different, and choosing one over the other can have significant impacts on the model's effectiveness, reliability, safety, and interpretation.

In general, the goal of AI is to build models that can automatically perceive, reason, comprehend, and learn from complex data or situations to react accordingly. This involves creating algorithms, machines, and techniques that enable decision-making. AI is a rapidly evolving field of research driven by advances in algorithms, computing power, data collection, and the availability of large-scale data sets. It draws heavily on concepts from statistics, probability theory, linear algebra, calculus, and optimization theory.

### 2.2.2 Rule-based Systems

In RBS, developers (i.e. people) define a set of rules and the logic for how a model makes decisions. For example, one could define a set of IF/ELSE rules and infer a decision based on properties: "IF IT RAINS, THEN TAKE THE CAR, ELSE GO BY FOOT. These rules can be convoluted and nested in any possible way, with different logical operations, and can be visualized in the form of decision trees (Figure 2.1):

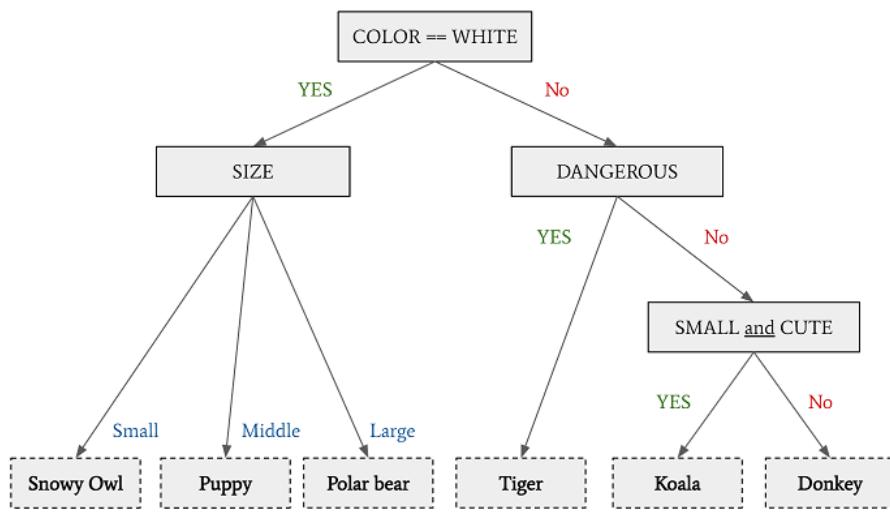


FIGURE 2.1: Example of a rule-based decision tree that classifies six different types of animals based on certain criteria.

### 2.2.3 Paradigms of Machine learning

In ML, models themselves figure out how to best achieve goals (usually minimizing a loss function; which is equivalent to maximizing a reward score) and self-sufficiently adapt and improve the model by learning from data; and so, the rules are *not* explicitly written or derived, but self-learned. Many sub-fields exist within

the area of ML, yet, the entire span can be categorized roughly into the following main ML paradigms:

1. **Unsupervised methods:** Do not require labeled data for training.
2. **Supervised methods:** Rely on labeled data for training and inference.
3. **Self-supervised methods:** Utilize the inherent structure within data to create labels for training.
4. **Reinforcement Learning methods:** Use trial-and-error to learn optimal actions in an environment.
5. **Hybrid methods:** Combine multiple techniques to achieve better performance in machine learning tasks.

## Unsupervised Learning

Unsupervised learning methods refer to types of algorithms that learn patterns from unlabeled data (Barlow, 1989; Ghahramani, 2003; Khanum et al., 2015). The most prominent unsupervised methods are for **clustering** (Ruspini, 1969; Xu and Wunsch, 2008; Madhulatha, 2012), **anomaly detection** (Chandola, Banerjee, and Kumar, 2009; Eskin et al., 2002; Schlegl et al., 2017), **Latent variable learning** (Everett, 2013; Valpola, 2015) and **graphical-models** (Newman, Watts, and Strogatz, 2002; D'Andrea, Ferri, and Grifoni, 2010), as well as for **dimensionality reduction** (Van Der Maaten, Postma, Herik, et al., 2009) and **association learning** (Cios et al., 2007). Since obtaining labeled data can be very tedious, difficult, costly, and slow, unsupervised methods often offer an attractive solution; but may be more challenging.

## Supervised Learning

Supervised methods (Caruana and Niculescu-Mizil, 2006; Bzdok, Krzywinski, and Altman, 2018; Muhammad and Yan, 2015) differ from unsupervised methods in that they require labeled (i.e. ground-truth) data, where each data point is already assigned a label. The goal is to train a model using this labeled data and subsequently employ the trained model to predict labels for new, previously unseen, and unlabeled data. In general, the success of any model in making accurate predictions on new data depends essentially on its ability to generalize the patterns learned from the labeled (training) data. Supervised learning has many applications across various domains (Tiwari, 2022) including classification, regression tasks, ranking, sequence labeling, semantic segmentation, and even image-to-text generation via large generative language models.

## Self-Supervised Learning

Self-supervised learning methods, also often referred to as "pretext"<sup>1</sup> tasks, can be thought of as a hybrid of supervised and unsupervised learning. In particular, self-supervised learning diverges from the supervised and unsupervised paradigm by generating its own (self-) supervised label (i.e. ground-truth) directly from the data itself; without the need for extrinsically labeled examples (the labels are e.g. parts

---

<sup>1</sup>The term *pretext* underscores that a given task serves as a preliminary step (other than the main objective), offering so a *pretext* for obtaining "useful" (Zhang, Isola, and Efros, 2017) representations.

of or embedded within the data itself). The main idea is to make use of the knowledge inherent (hidden) in vast amounts of unlabeled data by learning intermediate (strong, meaningful) representations that encode semantic or structural information useful for further downstream tasks. Being "useful" refers to a representation that is *adaptable* for various tasks, especially those unknown during the training phase (Zhang, Isola, and Efros, 2017). Overall, the key objective of self-supervised learning is to capture the underlying structure of raw data without explicitly relying on (extrinsic) labeled examples; it allows us to avoid the high cost of annotating large data sets, as we can extract useful information from unlabeled data; without having to label it beforehand (Weng, 2019; Jaiswal et al., 2021).

### Reinforcement Learning

In Reinforcement learning (RL) the idea is that *agents* in a certain *environment* - typically formulated as a Markov decision process (MDP) (Feinberg and Shwartz, 2012) - perform *actions* to maximize a long term, cumulative *reward* based on *observations* (Sutton and Barto, 2018; Kaelbling, Littman, and Moore, 1996; Li, 2017). The mapping of observations to actions itself is called a *policy*. Thereby, a balance between exploratory- and exploitative behavior must be established (Coggan, 2004; Thrun, 1992); i.e, how much should the agent try out new actions, strategies, etc. to improve the current policy and gain new knowledge, and how much of our current knowledge do we (confidently) want to exploit. The agent then continually optimizes its policy based on the rewards it obtains.

### Hybrid Learning Methods

Hybrid models can be seen as models that combine different approaches, such as **semi-supervised learning** (Chapelle, Scholkopf, and Zien, 2009; Zhu, 2005; Zhu and Goldberg, 2009) which uses labeled data as well as (a large amount of) unlabeled data. Another example of a hybrid approach is **semi-supervised reinforcement learning** (Finn et al., 2016) where a reward function can e.g. only be evaluated using labeled Markov decision processes (MDPs) and an agent must generalize its policy to states it might encounter in a set of unlabeled MDPs. Other relevant strategies in hybrid methods include **unsupervised pretraining + supervised fine-tuning** and **active learning + supervised learning**, both of which hold considerable importance in natural language processing. Additionally, **transfer learning** is also an example of a flexible strategy that can integrate elements of both learning paradigms: supervised (fine-tuning with labeled data on a particular task) and unsupervised (pretraining on an unlabeled large data set or corpus).

#### 2.2.4 Weakly-Supervised Learning

Supervised learning approaches use a large number of training samples to build prediction models. However, since the data-labeling procedure is expensive, it is often difficult to obtain accurate supervision information, such as exact ground-truths. To address this challenge, machine-learning algorithms with "weak supervision" are used. Weak supervision refers to a set of techniques that require less supervision and are split into three main areas (Zhou, 2018): incomplete supervision, where only a portion of the training data is labeled; inexact supervision, where only coarse or meta labels are provided; and incorrect supervision, where the given labels are not necessarily ground-truth.

### 2.2.5 Challenges in Unsupervised Clustering Methods

Unsupervised clustering algorithms face an inherent problem when clustering/-grouping based on similarity measures. This is due to the model's perception or interpretation of the term "similarity" differing from the expected one (e.g. of a developer). Consider Figure 2.2 for example, where we group images of digits from the MNIST data set in an unsupervised manner, i.e. without labels. But how does the model know what we're looking for? If, for example, the model computes similarity based on pixel values, it will not find clusters based on parity. Similarly, if it clusters based on parity, it will fail to cluster the images based on the pixel values. Even if the model were to cluster the images based on both, parity and pixel values, it would still fail the task if the true purpose was to cluster based on common factors (i.e. denominators). This is important to remember and must be handled explicitly, which may need a semi-supervised learning strategy and hence a shift away from entirely unsupervised learning. Overall, performing clustering on "something" is always feasible; the real question is rather "how beneficial" it is.

Therefore, unsupervised clustering encounters difficulties in determining similarity without clear guidance, often leading to divergence from human expectations. To address this, one can try to more explicitly define the clustering task or incorporate human expertise and domain knowledge. Other techniques such as feature engineering, representation learning, and dimensionality reduction can also aid in aligning the model's perception of similarity with the desired results. Additionally, iterative clustering refinement and the evaluation of the practical utility of clusters may prove necessary or helpful, while interpretability-enhancing methods can provide more transparent insights and explanations.

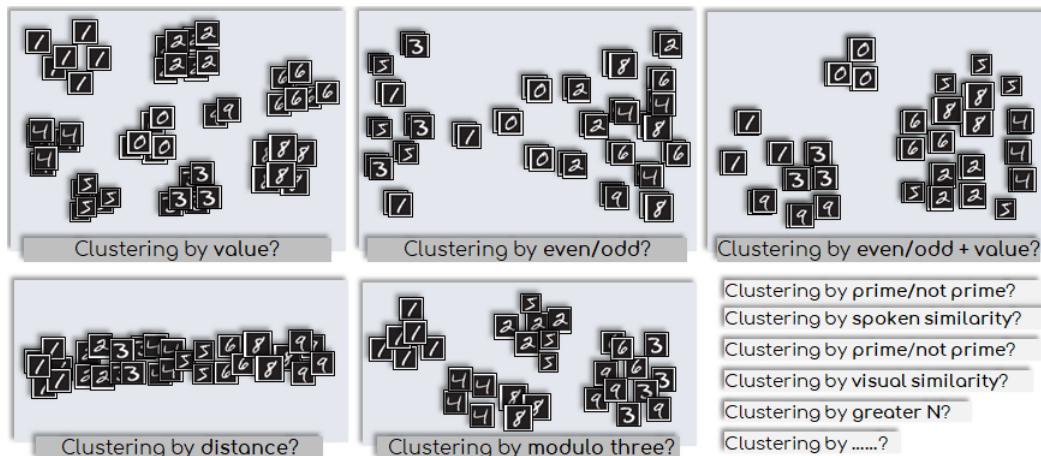


FIGURE 2.2: Demonstrating clustering ambiguity and diverse clustering combinations in unsupervised settings [using the MNIST data set (Mu and Gilmer, 2019)].

### 2.2.6 Rule-based vs. Machine Learning

Although Rule-based Systems (RBS) and Machine Learning (ML) are both important key concepts in AI, they have significant differences that lead to decisive consequences. While ML was developed due to the many intrinsic limitations of RBSs, each approach has its own pros and cons and different purposes and applications.

To start with RBSs, they have a clear advantage in that they are easy to understand, comprehend, and retrace. In fact, anyone with no background in computer science can easily grasp the concept of rule-based systems. For instance, in a decision tree, given any input and its corresponding output, the reason why that input yielded its corresponding output can always be traced back simply by looking at the path it took within the tree. This feature of RBSs provides a strong advantage due to their implementation, explainability, and interpretability. Safety can be ensured in RBSs by controlling their outputs (i.e. decisions) and checking all possible paths.

However, there is one major and critical problem: **Manually devising RBSs is not scalable, complex, and erroneous.** Imagine that we have a data set of various vehicle defects and have already built a decision tree that can determine the precise cause of each defect. But, what happens when we want to add new defects to the existing tree? In this case, we would need an expert who has domain knowledge and can help us to build new rules. However, this process can be quite tedious and laborious, and it can be a challenge to ensure that we cover all possible causes and keep track of all exceptions. Also, what if there are more individual exceptions than generalizable rules? What if we forget a rule or if the causes of previous defects change due to upgrades or modifications in the vehicle manufacturing process? Moreover, what if the rules themselves change faster than we can update them (e.g. recall the six Vs; Chapter 2.1)?

Consider now a scenario where we aim to categorize defects based on comments obtained from a user source. Specifically, we consider the following comment:

*"The defect is due to damage to the engine."*

One might consider the naive trivial rule, if the words `[damage ∧ engine]` occur, then classify the defect as "damaged engine". Now let's consider the following comment:

*"The defect is **not** due to damage to the engine."*

We, therefore, might naively modify our rule accordingly:

`[damage ∧ engine ∧ ¬ not]`

Next, we receive the following comment:

*"The defect **isn't** due to damage to the engine."*

... and we naively modify our rule accordingly:

`[damage ∧ engine ∧ ¬ not ∧ ¬ isn't]`

... but next, we receive the following:

*"The defect **wasn't** due to damage to the engine."*

... and adapt by naively updating the rules:

`[damage ∧ engine ∧ ¬ not ∧ ¬ isn't ∧ ¬ wasn't]`

... and next, we receive the following new input:

*No damage was found on the engine."*

... and adapt by naively updating the rules:

`[damage ∧ engine ∧ ¬ not ∧ ¬ isn't ∧ ¬ wasn't ∧ ¬ no]`

... and next, we receive the following new input:

*"The defect is due to the **harmed** engine."*

... and adapt by naively updating the rules:

`[(harm ∨ damage) ∧ engine ∧ ¬ not ∧ ¬ isn't ∧ ¬ wasn't ∧ ¬ no]`

... and next, we receive the following new input:

*"Engine damaged."*

... and adapt by naively updating the rules:

`[(harm ∨ damage ∨ damaged) ∧ engine ∧ ¬ not ∧ ¬ isn't ∧ ¬ wasn't ∧ ¬ no]`

But even after modifying and expanding our rules to fit all the above situations, countless exceptions would still remain unaccounted for and incorrect, such as:

*"The defect **might** be due to damage to the engine."*

*"Engine totally fine, but tank **damaged**"*

*"Engine is damaged, **no** other part were harmed."*

*"I check the engine, the tank, the windows and all was **damaged**."*

One quickly and easily sees that **such an approach is not scalable, practical, or even sensible**. This becomes even more evident when trying to apply it to more complex data such as images, videos, or audio. Thus, RBS suffers from significant drawbacks such as being inflexible, incapable of handling unknown scenarios, requiring extensive time and resources for rule creation and maintenance, having limited understanding and potential inaccuracies, and carrying the risk of bias due to questionable assumptions or incomplete data.

ML on the other hand, handles this scenario differently using probabilities, statistics, and mathematics in general. Instead of trying to completely mimic an expert's decision process, ML merely either learns based on the results of the experts (e.g. supervised learning with labeled data) or from extracting patterns of raw or prepossessed data (e.g. unsupervised learning), where outputs or predictions are inferred relative to combinations of input variables and parameters. Also, unlike RBS, if we were to generate a new model based on new data (i.e. vehicles and defects), we would not have to repeat the process of generating the rules, but would simply re-run the ML algorithm to either re-train or fine-tune on the new specific data set. However, ML methods need a large set of data to effectively train and achieve satisfying performance. Additionally, ML methods come at the expense of explainability and interpretability and there is still heavy research and projects devoted to this problem (Molnar, 2020; Du, Liu, and Hu, 2019; Samek, Wiegand, and Müller, 2017; Burkart and Huber, 2021; Bhatt et al., 2020), i.e, given an input and its corresponding output, answering *"why does that output correspond to that input"* remains an active and open challenge.

A visual summary illustration of the primary paradigms that are key in Artificial Intelligence is given in Figure 2.3 below.

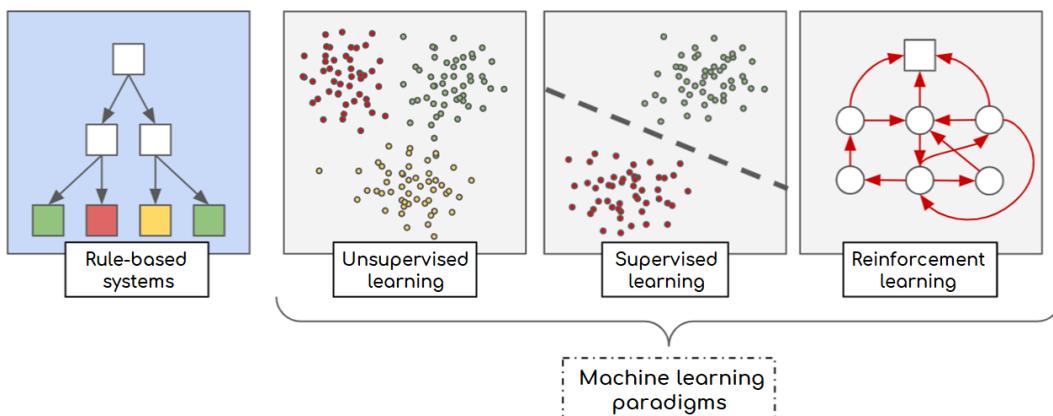


FIGURE 2.3: A graphical illustration of the main paradigms in Artificial Intelligence.

### 2.2.7 Model Generalization

We are interested in training a model that generalizes well to new and unseen data. This is to ensure its practical applicability and reliability across various (new) scenarios. A common technique is to split all our available data into three disjoint data sets: **Training**-, **Test**-, and **Validation** data. One then *trains* the model on the *training set*, but then *tests* its effectiveness (e.g. accuracy, precision, recall, etc.) on the *test set*. The purpose of the *validation set* is to try out different hyper-parameters, i.e. parameters that are not learned by the model but pre-fixed by the developer.

One must use a separate data set for each purpose as otherwise we risk over-adapting the configuration of the hyper-parameters on our data; known by the name of **over-fitting**. The opposite, i.e. when the data model fails to learn the training data, is similarly referred to as **under-fitting** (Koehrsen, 2018). Over- and under-fitting are general terms and, therefore, not exclusively bound to any specific type of learning algorithms, but should always be avoided when trying to train and define a generalized model. Yet, there exist multiple different ways to counteract both problems (Browne, 2000; Jabbar and Khan, 2015; Aalst et al., 2010).

#### Take-Away

AI may be defined as the development of intelligent machines that can perceive, reason, and take actions based on goals. Rule-based methods rely on explicitly defined rules, while Machine Learning methods learn patterns from data. Unsupervised clustering can be quite tricky to get it right. Weakly supervised learning aims to learn from noisy or imperfect labels. Model generalization refers to the ability of a model to perform well on unseen data.

#### Keywords

Machine Learning • Clustering • Weak Supervision • Model Generalization.

## 2.3 Data Encoding & Dimensionality

For "Artificial Intelligence" to be applied to data, especially in Machine Learning, raw data has to be first transformed into a format that can be used for further processing, i.e. training and prediction. *Data encoding* is the process of converting data from one form to another. The input encoding is typically done in the form of a numeric input vector encoding. The term *dimensionality* refers to the number (dimensions) of feature input variables; e.g. the size (i.e. dimension) of the encoding vector. Overall, there exist multiple different approaches to pre-processing and *encoding data* depending on the type of the data. Typically, one distinguishes between the following different main types of data:

1. **Numerical data** refers to quantitative values that can be either discrete (whole numbers) or continuous (decimals). Ensuring consistent scales and addressing missing values are important challenges when encoding numerical data.
  - Discrete (e.g.  $x \in \mathbb{Z}$ )
  - Continuous (e.g.  $x \in \mathbb{R}$ )

2. **Categorical data** classifies variables into different categories. Binary data has two categories, ordinal data has a specific order, and nominal data has unordered categories. Choosing the appropriate encoding method for categorical data based on its type and number of categories is a challenge.
  - Binary (*e.g. ON, OFF*)
  - Ordinal (*e.g. BAD, ALRIGHT, GOOD, AMAZING, ...*)
  - Nominal (*e.g. GERMANY, SPAIN, USA, CHINA, FRANCE, ...*)
3. **Text data** is unstructured information in the form of sentences, paragraphs, or documents. Tokenization, removing special characters and stop words, handling large textual documents, and capturing word semantics with accurate methods are challenges in effectively encoding text data.
  - Natural language (*e.g. "Hello world!"*)
  - Coding language (*e.g. "private int x = 42;"*)
  - ...
4. **Time series** data records values at different points in time where the order is of great significance. Challenges in encoding time series data include handling trends, missing observations, irregular time points, and selecting appropriate models for analysis or forecasting.
  - Cyclic (*Values that repeat at regular intervals; e.g., seconds, minutes, hours, days, weeks, months, quarters, seasons, etc.*)
  - Non-Cyclic (*Values that lack regular repetition; e.g., specific years, decades, centuries, time intervals, etc.*)
5. **Audio, Images, and Video** data represent sensory information and require specific pre-processing methods for encoding. Challenges include transforming audio signals into (semantic) numerical representations, handling variations in formats or quality, and transforming pixels into numerical arrays or embedding representations for images and videos.
6. **Graphs & Networks** represent relationships between entities (e.g. social networks, protein-protein interactions, etc.). Challenges in encoding graph data involve determining the appropriate representation, handling missing or noisy data, and dealing with differently-sized graphs.
7. **Etc. ...**

Especially in the case of non-numeric values, it is not trivial to define what a "good" encoding scheme is as it depends on multiple factors, especially its use case and actual application. Overall, a "good" encoding is any encoding that provides maximum utility and effectiveness for its (specific) purpose, which can, for example, be characterized by considering the following questions:

- "How accurately can we reconstruct the original input from its encoding?"
- "Do similar inputs yield similar encodings, while dissimilar inputs result in distinct encodings?"
- "Is the encoding minimal, i.e. can we achieve comparable model effectiveness with a smaller encoding?"

- "What is the speed and efficiency of generating the encoding?"
- "How well does the encoding scheme scale?"
- "Does the encoding maintain relationships, associations, and correlations within the data?"
- "Is the encoding an injective function?"
- "Does the encoding remain robust against noise or data perturbations?"
- "Does the encoding contribute to interpretability or provide meaningful insights into the data?"
- "Are there any inherent biases or limitations in the encoding that might affect its performance or applicability?"
- "Can the encoding preserve privacy or anonymize sensitive information when handling such data?"
- Etc. ...

It is further important to be aware of the differences and relationships between data *encoding*, data *compression*, data *embedding* (Hotho, Nürnberg, and Paaß, 2005; Almeida and Xexéo, 2019; Tian, 2003) and *dimensionality reduction* (Lee and Verleysen, 2007; Van Der Maaten, Postma, Herik, et al., 2009; Tenenbaum, De Silva, and Langford, 2000) and we hereon encourage the interested reader to review the respective parts in literature. But overall, the following differences hold: (i) a data encoding transforms data into a different representation, (ii) a compression reduces size while preserving information, and (iii) an embedding maps data into a lower-dimensional space while maintaining its inherent, semantic structure or relationships.

Hence, both data compression and data embeddings are ways of encoding information (they serve different purposes and utilize different techniques), but not all data encodings involve compression or embeddings. Given a fixed (initial) encoding, performing dimensionality reduction can be seen as lowering the number of dimensions for that given encoding; e.g. with the goal to encode similar data into similar (new, lower-dimensional) data representations (i.e. encodings, embeddings) that are smaller than the corresponding original representations. Compression can be lossy or lossless; and encodes information "smaller" than the original representation, however, the main goal is to reconstruct the exact original data as lossless and accurately as possible. Therefore, dimensionality reduction can be seen as a form of *lossy* compression, but compression is not the same as dimensionality reduction (both decrease data volume, but: dimensionality reduction prioritizes preserving utility for specific tasks; compression primarily aims at minimizing size without information loss).

Embeddings, on the other hand, follow a different concept and were (first) typically used in conjunction with words, sentences, documents, etc. in natural language processing (NLP); where each word is "embedded" (i.e. encoded) in a vector space, such that words that are closer in the space denote a similar **meaning**. Commonly, two words are considered similar if they occur in similar *contexts* (Goldberg and Levy, 2014), rather than solely co-occurring frequently, as seen in earlier methods like GLOVE (Pennington, Socher, and Manning, 2014), which relied on co-occurrence statistics within a corpus to generate word representations. However, co-occurrence-based approaches lacked the robustness needed to accurately learn

meaningful word similarities due to limitations in contextual comprehension, polysemy resolution, and adaptability across various tasks.

There exist several hypotheses in the literature that address assumptions on what a "good" embedding is (Ouali, Hudelot, and Tami, 2020):

- **Smoothness Assumption:** Data points that are close together in the feature space should have labels or meanings that are equal or similar.
- **Cluster Assumption:** Dense regions of data points form clusters and points within the same clusters should hold the same label or meaning.
- **Low-density Separation Assumption:** Decision boundaries of classes should be located in low-density (sparse) regions.
- **Manifold Assumption:** High-dimensional data can be represented on a low-dimensional manifold.

Regardless of the transformation scheme used, the output is always an encoded data instance with a certain dimensionality. In the next section, we will discuss the implications of encodings with large dimensions and why dimensionality matters.

### Take-Away

When preparing data for Artificial Intelligence (AI), it is crucial to carefully consider the data encoding method and its dimensionality. Feature encoding methods vary depending on the type of data being processed, such as numerical, categorical, text, time series, audio, images, graphs, etc. The effectiveness of data representations can depend on factors like information loss, quality of reconstruction, similarity or semantic preservation, efficiency, scalability, consideration of contextual relationships, and the actual use case.

### Keywords

Data Preparation • Pre-Processing • Encoding • Dimensionality

## 2.4 The Curse & Blessing of Dimensionality

*"The curse and blessing of dimensionality"* is a common phrase used in mathematical sciences, but particularly in ML, that is concerned with the benefits and issues arising when dealing with high-dimensional functions or data (i.e. encodings). High-dimensionality can be seen as a **curse** (Bellman, 1961; Köppen, 2000; Indyk and Motwani, 1998) or, contrarily, as a **blessing** (Gorban and Tyukin, 2018; Anderson et al., 2014; Kainen, 1997). However, it can also be acknowledged as both, a curse and blessing (Donoho et al., 2000) and therefore plays an important role when developing any ML model.

### 2.4.1 The Curse of Dimensionality

The curse of dimensionality, first introduced by Bellman, 1961, refers to issues arising in high-dimensional spaces that do not occur in low-dimensional spaces. The curse refers to situations in which the number of data points is significantly smaller compared to the actual dimensionality of the data. As the number of dimensions

increases, the volume of the space also expands rapidly, making the data points sparser. To achieve reliable outcomes, exponentially more data is required as the dimensionality increases.

It can also refer to the perceived difficulty in systematically exploring a complex, multi-dimensional space, accurately estimating a general function within that space, integrating functions across multiple dimensions, and challenges associated with navigating and analyzing high-dimensional data or scenarios (Donoho et al., 2000).

In the following, we will introduce some of the many notorious and common issues that arise when dealing with high-dimensional spaces.

### Distance Concentration

Distance metrics play an important role in ML applications as they allow for measuring the similarity and dissimilarity of data instances and can be defined in a multitude of different ways (Li and Li, 2010). However, such distance metrics suffer from a phenomenon called *distance concentration* that occurs in high dimensional spaces whereas dimensionality grows, so do the distance values, consequently lowering the contrast provided by such typical distance metrics (Beyer et al., 1999). I.e, the Euclidean distance for two given random vectors  $\vec{v}_{(d)}, \vec{w}_{(d)}$  of dimension  $d$  with independent and identically distributed variables increases the higher  $d$  is (as there are more variables involved):

$$\|\vec{v}_{(d)} - \vec{w}_{(d)}\|_2 = \sqrt{\sum_{i=1}^d (v_{(d)}^{[i]} - w_{(d)}^{[i]})^2} \quad (2.1)$$

In other words, as the number of dimensions in a vector space increases, the distances between points in that space also increase, up to a point where the maximum distance between a reference point and any list of randomly sampled points converges (in expectation) against its minimum distance to that reference point (Aggarwal, Hinneburg, and Keim, 2001); meaning the smallest distance approaches (or equals) the largest distance:

$$\lim_{d \rightarrow \infty} \mathbb{E} \left[ \frac{dist_{(d)}^{\max} - dist_{(d)}^{\min}}{dist_{(d)}^{\min}} \right] = 0 \iff \lim_{d \rightarrow \infty} \mathbb{E} \left[ \frac{dist_{(d)}^{\min}}{dist_{(d)}^{\max}} \right] = 1 \quad (2.2)$$

Therefore, the concept of neighborhoods (e.g. k-nearest-neighbours, k-Means, etc.) becomes unstable due to the distance between the nearest and furthest neighbor collapsing; which is a serious concern. Further, it holds that any two i.i.d. randomly drawn vectors tend to be mutually orthogonal to each other, i.e. that:

$$\lim_{d \rightarrow \infty} \vec{v}_{(d)}^T \vec{w}_{(d)} = 0 \quad (2.3)$$

### Data Sparsity

Let us assume a  $d$ -dimensional space partitioned into equal subsections of size  $m > 0$  within a valid interval range of size  $r$ . The number of cells  $n$  increases exponentially with  $n = (r/m)^d$ . Suppose further  $k$  points are randomly placed in such  $d$ -dimensional space. The lower  $d$  is, the fewer empty partitions exist; with more points per partition. The higher  $d$  is, however, the more empty partitions emerge, with each non-empty partition having crucially fewer points in it. In fact, given a fixed  $k$ , augmenting  $d$  rapidly leads to the number of partitions clearly exceeding

that of the points; and consequently many empty partitions (Figure 2.4). A naive measure of data sparsity is the ratio of points to cells  $\rho = \frac{n}{k} = \frac{(r/m)^d}{k}$ .

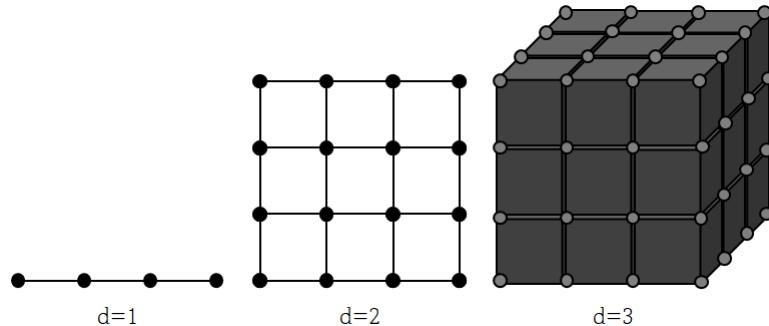


FIGURE 2.4: Data Sparsity: Illustration of the partitioning of dimensional spaces into cells.

## Combinatorics

Assuming the simplest case of binary variables, i.e. variables with just two possible values, given  $d$  dimensions the number of possible combinations rapidly explodes to  $2^d$  and is known under the name of *combinatorial explosion* (Schuster, 2000).

## Other

Many more problems, challenges, and complexities arise when dealing with high-dimensional data, especially in anomaly detection, machine learning, hypothesis reasoning, optimization, and numerical integration; but also concerning computational limitations and bottlenecks, visualization, interpretability and explainability, reduced sample efficiency, overfitting, and many more.

### 2.4.2 The Blessing of Dimensionality

The blessing of dimensionality, on the other hand, is concerned about the benefits such high-dimensional settings enable, but are less popular in comparison. These include increased information capacity, enhanced discriminative power, and the ability to transform some non-convex problems into convex ones, as well as some non-linear problems into linear decidable problems in higher dimensions (such as the notorious XOR problem). Moreover, these advantages can extend to diverse applications, such as measuring phenomena within the geometry of Banach spaces, where random fluctuations can be well controlled in very high dimensions, as well as the success of asymptotic methods which enable statements to be made where "moderate dimensions would be too complicated." (Donoho et al., 2000).

**Take-Away**

Machine Learning (ML) faces challenges due to high dimensionality such as concentrated distances, data sparsity, and complexities in data handling. However, it also provides benefits like increased information capacity and improved discriminative power. It is crucial to understand these dynamics to develop effective ML models and manage data efficiently. Ultimately, there is a clear **underlying compromise between complexity and simplicity**.

**Keywords**

Machine Learning • High Dimensionality • Challenges • Compromise

## 2.5 Data Augmentation

Usually, the amount of actual available and accessible (labeled) data is limited and often insufficient for performing effective ML, i.e. the overall final performance of a given ML model is not only dependent on the quality of the data it is being trained upon but also crucially on the size of the respective data set (Barbedo, 2018). Gathering and obtaining such data is generally not only challenging and time-consuming, but also error-prone, and one of the reasons why rule-based methods are still commonly applied instead of ML approaches; namely due to deficient data sets.

*Data augmentation* (Van Dyk and Meng, 2001; Zhong et al., 2020; Wong et al., 2016; Perez and Wang, 2017) refers to synthetically generating new (similar, related) data instances from a given limited data set, thereby essentially increasing the size (entropy) of the data set. Data augmentation also helps to counteract over-fitting on ML models and is typically a non-complex procedure that can be quickly and easily implemented and integrated; especially without modifying the existing model architecture, e.g. by adding slight modifications or perturbations to already existing (data) instances, applying diverse (strong) augmentations, or combining different instances in different (stochastic) ways.

Depending on the type of data, different data augmentation strategies exist. For instance, augmenting, adding noise, or manipulating text data varies fundamentally from doing the same for image data. Similarly, the augmentation techniques for graph networks differ from those used for audio or numerical data. It's important to note that data augmentation might not always be beneficial, and excessively strong data manipulations can actually hinder the learning process. This is because, although appropriate image data augmentation improves model robustness by exposing it to diverse variations, excessive modifications may ultimately confuse the model, reducing its ability to generalize and perform well on unseen data. Finding the right balance is crucial for effective machine learning.

### 2.5.1 Assessing Generated Data Quality

A question remains: What is the right data augmentation, and how can we evaluate its quality? This is important because incorrect data augmentation can harm training effectiveness, result in longer training time, and add unnecessary complexity.

In their work "Affinity and Diversity: Understanding Data Augmentation Mechanisms" (Gontijo-Lopes et al., 2020) the authors go into the fundamental aspects

of data augmentation and introduce two "interpretable and easy-to-compute measures": *Affinity* and *Diversity*.

"Affinity" measures how much an augmentation alters the learned data distribution by a model, while "Diversity" gauges the complexity of the augmented data concerning the model and learning process. The authors report that neither measure singularly predicts the effectiveness of data augmentation; but instead, optimizing both Affinity and Diversity together influences performance.

### Take-Away

Data scarcity in Machine Learning can be addressed by data augmentation. This involves generating additional instances of similar data to enhance model performance. However, excessive or unrealistic modifications from augmentation can hinder a model's ability to generalize. "Affinity" and "Diversity" are intuitive quality metrics.

### Keywords

Data scarcity • Data Augmentation • Model Performance • Generalization

## 2.6 Optimization

*Optimization* plays a crucially important role in mathematics and any related field (Rao, 2019; Chong and Zak, 2004), and, therefore, consequently establishes the fundamental back-bone of many ML algorithms (Sra, Nowozin, and Wright, 2012; Sun et al., 2019), but also for many more applications. The idea of *optimization problems* is to minimize a given **objective function** subject to **constraints**. Commonly, one defines such objective function as a *loss*-function minimization problem. Minimizing a "loss" function can be viewed as the equivalent of maximizing a "score" function (e.g. simply by multiplying by minus one). In fact, the key to training any ML model is always to optimize the model in such a way that it performs "optimal" in regards to pre-defined (usually fixed) loss functions. The more complex the loss function is, the harder or more infeasible and intractable it becomes to solve the problem "exactly" (numerically). Additionally, the loss function is often even defined without concrete knowledge about the implicit *loss-space* and it is up to the optimization algorithm to "figure out" how to traverse the search space or, generally speaking, how to minimize the loss.

*Convex*-Optimization (Boyd, Boyd, and Vandenberghe, 2004; Ben-Tal and Nemirovski, 1998) is a sub-field of optimization that particularly focuses on convex functions (Roberts, 1993), i.e. continuous function where the value at the midpoint of any two (arbitrary but fixed) points is less than or equal to the mean of the values of those points. Another way to think of convex functions is functions that do not exceed the segment line between any two function values. Figure 2.5 provides an example for both, a convex and non-convex function, on a one-dimensional input function. The contrary to convex functions is called *concave* functions. Convex or concave problems are regarded as easy to solve (both, theoretically and practically).

However, most problems we encounter are generally high dimensional, very complex, and especially non-convex. Still, they often comprise properties of convex functions near local minima or maxima which might be exploited. Non-convex

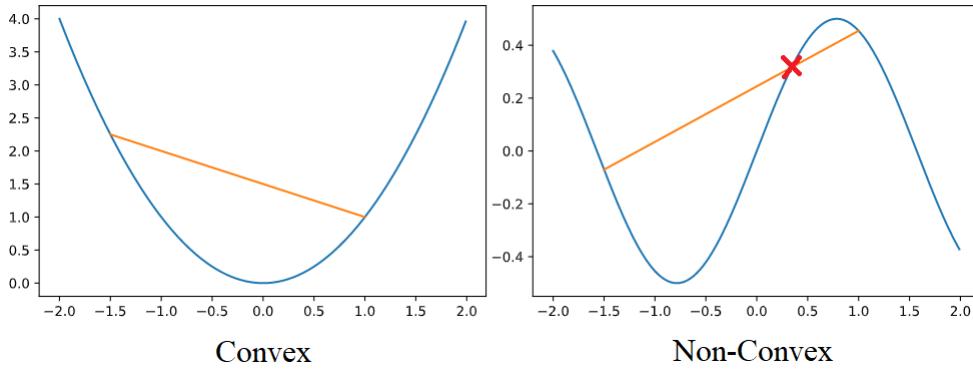


FIGURE 2.5: Visualizing two simple one-dimensional input convex and non-convex functions.

problems present challenges because they can have multiple local minima or maxima, making it difficult to determine the global optimum. Escaping from local to global optima in optimization tasks can e.g. be achieved through the utilization of diverse optimization techniques, including heuristics, or stochastic approaches.

Generally, we are interested in finding or approximating global minima, since that is where the best possible solution configuration lies for any given loss function. However, such global minima must not be unique. Multiple global minima could exist within a problem space and provide different optimal solutions, e.g. revealing the diversity and richness of potential configurations within given constraints and objectives.

In the following subsections, we will introduce the prominent and very powerful approach to solving optimization problems, called *gradient descent* (GD). We will further introduce the concept of *back-propagation* that is typically used to perform GD for solving optimizations. GD, however, is used for minimization problems, the analogy for maximization problems is called *gradient ascend*.

### 2.6.1 Gradient Descent

*Gradient Descent* (GD) is an iterative type of optimization algorithm for finding a (local) minimum within a differentiable function by repeatedly taking steps (*descending*) in the negative direction of the gradient on a given point (Ruder, 2016; Qian, 1999; Andrychowicz et al., 2016). GD works for spaces of any number of finite and infinite dimensions and can be applied to a system of linear and non-linear equations. When the loss-space is convex it converges towards a global solution, since then any local minimum is also a global minimum. Since real-world problems are seldom convex, different techniques, variations and methods exist to try converging to a mostly global solution (Bottou, 2010; Du et al., 2019; Bottou, 2012) In its most simple form, GD can be described by the following formula:

$$\vec{v}_{n+1} := \vec{v}_n - \eta \nabla L(\vec{v}_n) \quad (2.4)$$

where  $\nabla L$  is the gradient of the loss function  $L$  and  $\eta$  the *learning rate* (the step-size), and  $v_n$  represents a parameter configuration at time-step  $n$ . In other words, the idea is the following: Since the optimal solution lies on a minimum, given our current position in a loss-space, we descent along the direction of the **steepest descent** until our gradient is zero, which would imply we reached our optimal solution.

Another way of imagining gradient descent is that of a downwards-rolling ball in a bumpy field. The obvious challenge here is *not* to get trapped in local minima. Again, local minima do not necessarily represent optimal or even "good" solutions. In fact, solutions found at local minima can be extremely bad, even though they might be relatively "close" to the position of the global minima within the search space. The output of classical GD, especially the iterative path it traverses, is very much dependent on the initial starting point, however, different starting points could, especially in the case of a unique global solution on convex cases, lead to the same solution. In other words, the trajectory and ultimate solution of gradient descent (which also relies on the optimization algorithm) are not inherently unique. An illustration of the iterative gradient step path is given in Figure 2.6.

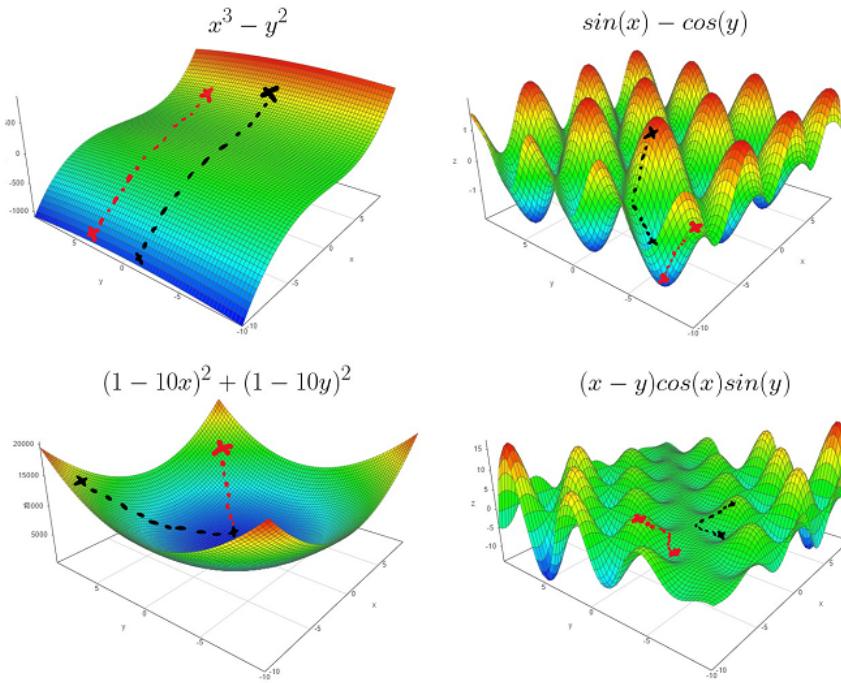


FIGURE 2.6: Visualizing the iterative step-path of gradient descent on four different loss spaces for two different starting points.

There are numerous approaches and techniques aimed at preventing GD from converging towards suboptimal local solutions. These methods include employing diverse sampling algorithms (Yazan and Talu, 2017), incorporating momentum (Qian, 1999), or utilizing adaptive optimizers like Adam (Kingma and Ba, 2014); and its very effective, powerful and renowned variant AdamW (Loshchilov and Hutter, 2017). However, interestingly, it has also been observed that, for many real-world problems and random loss functions in Deep Neural Networks, converging towards local minima can (but does not necessarily) achieve similar performance than one would obtain at global minima, or seem to be not affected by *bad* local minima; that is, that many search spaces consist of many approximately equally good sub-solutions compared to the optimal solution or finding a global minimum may not be necessary and may even reduce overfitting (Swirszcz, Czarnecki, and Pascanu, 2016; Choromanska et al., 2015). In particular, Choromanska et al., 2015 show that for random loss functions in large-size decoupled networks, the lowest critical values of a random loss function form a layered structure and they are located in a

well-defined band lower-bounded by the global minimum, and that the number of local minima outside that band diminishes exponentially with the size of the NN.

Moreover, many local minima are situated on saddle points and, thus, contain sub-dimensional paths to escape such local optima (Dauphin et al., 2014). In fact, particularly for neural networks it has been further observed that optimal solutions are not isolated points, but instead connected through paths, i.e. simple curves of near-constant loss. (Draxler et al., 2018; Izmailov et al., 2018).

### 2.6.2 Back-Propagation

*Back-Propagation* (BP) is an effective and widely used technique for calculating the gradients of loss functions. This method has proven to be very successful and has become the most popular and standard approach for optimization tasks, particularly in the context of artificial neural networks (Rumelhart et al., 1995; Riedmiller and Braun, 1993; LeCun et al., 1989). Additionally, the idea of BP itself invoked interest and speculation in whether or not it may offer insights for understanding learning in the cortex (Lillicrap and Santoro, 2019; Lillicrap et al., 2020). BP computes the gradient of a loss function relative to trainable parameters (i.e. weights) and is often used in conjunction with *automatic differentiation* (Paszke et al., 2017).

For example, suppose we have a composite loss function  $L$  which aggregates multiple sub-functions  $f_i$  together, based on an input  $x$ . We are now interested in adapting the weight parameter  $w_j$  at function  $f_j$ , i.e. for  $L \triangleq f_i(f_{i-1}(\dots f_1(f_0(x))))$ ;  $i > j \geq 0$ . We now look for the gradient  $\frac{\partial L}{\partial w_j}$  which we can calculate by applying the (backward) chain rule:

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial f_i} \cdot \frac{\partial f_i}{\partial f_{i-1}} \cdot \frac{\partial f_{i-1}}{\partial f_{i-2}} \cdot \dots \cdot \frac{\partial f_{j+1}}{\partial f_j} \cdot \frac{\partial f_j}{\partial w_j} \quad (2.5)$$

and, therefore, perform the gradient update with:

$$w_j := w_j - \eta \frac{\partial L}{\partial w_j} \quad (2.6)$$

Therefore, BP serves as a method to compute the gradient of a model's loss function relative to its parameter weights. Throughout the training, BP propagates the loss error backward through the model, calculating the gradient of the loss function for each parameter weight using calculus principles such as the chain rule. Optimizers such as AdamW leverage these gradients derived from BP to adjust the weights during BP, aiming to minimize the loss function and thereby enhance the model's performance; using GD. These optimizers update the weights by considering the gradient information obtained through the BP process.

In summary, while BP determines the gradients crucial for optimizing models, optimizers like AdamW utilize these gradients to modify the models' weights in a manner that reduces the loss function via GD.

**Take-Away**

Optimization, a fundamental concept in mathematics, extends across various disciplines; particularly Machine Learning (ML). It involves minimizing a loss function within defined constraints. In ML, this includes training models using methods such as Gradient Descent (GD) to navigate intricate terrains and minimize predefined loss functions. While convex optimization offers straightforward solutions, real-world problems often involve non-convex functions, posing greater challenges.

**Keywords**

Optimization • Loss Function • Gradient Descent • Convex Optimization

## 2.7 Neural Networks

The entire span of *Neural Networks* (NNs), together with all its very many variations (Schmidhuber, 2015; Gurney, 2018; Weng, 2017; Albawi, Mohammed, and Al-Zawi, 2017; Xu et al., 2018) and applications (Vellido, Lisboa, and Vaughan, 1999; Hunt et al., 1992; Zhang, 2000), goes far beyond the scope and feasible boundaries of this thesis. Yet we will give a short high- to low-level introduction to what an artificial NN is and how it works; since it is essential to following along this doctoral thesis and comprehending the fundamentals of modern AI.

Artificial NNs are inspired by the behavior of the human brain, specifically, the interconnection of neurons and information transfer. The goal is to recognize patterns and, thus, solve given tasks (i.e. minimizing an objective function: the loss function). Neurons in the human brain are inter-connected with each other by *synapses* which allows the exchange and flow of information (Suzuki, 2013).

On an abstract high-level, NNs mimic such behavior: they receive an input, propagate it toward different neurons, and produce an output (Figure 2.7). Each *neuron* is thereby connected with all the outputs from the previous layer (or the input layer). *Connections* establish (trainable) *weights* that are applied to each input respectively.

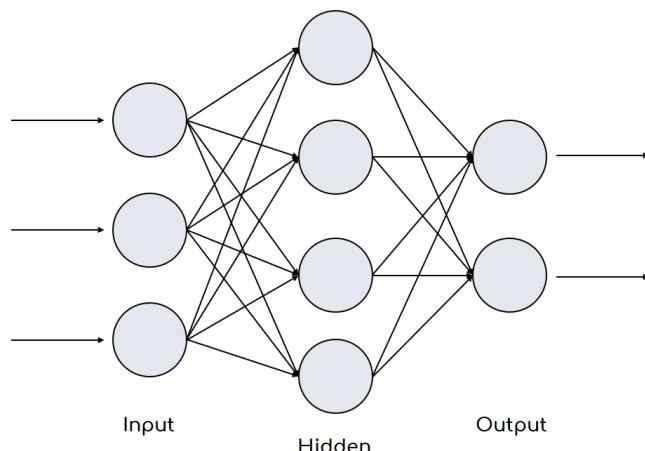


FIGURE 2.7: Depicting a simple one-layer neural network with three input- and two output values (without bias neuron).

Further, artificial NNs can be seen as modular blocks that can be arbitrarily stacked or combined in any creative way. That is, the output of a NN can function as the input for another NN, block, or neuron (e.g. Figure 2.8). The term *Deep Learning* (DL), thereby, refers to any NN with more than one hidden layer. A single-layered NN is also called a *single layer perceptron*, whereas similarly, a deep NN (DNN) with more than one hidden layer is referred to as *multilayer perceptron*. An NN with just an input layer, one hidden layer, and an output layer is not classified as a DL model; it's more precisely labeled as a "shallow" or single-layer NN. Correspondingly, a "narrow" NN denotes a specific type of NN characterized by fewer neurons or nodes within its hidden layers when compared to wider networks.

There is one very important component of NNs, namely the *activation function*. The purpose of activation functions is to **break the linearity** between layers. Activation functions are fundamentally important and transform the output of a neuron by applying non-linear functions (i.e. mappings) to the neuron's output to generate the actual neuron's (new) output. Omitting activation function results in a DNN collapsing into a single-layered perceptron. I.e., without non-linear activation functions, a multi-layer perception network is equivalent to a single perceptron network with no hidden layer and the same number of input and output neurons: it essentially collapses into a linear regression model (the network as a whole, regardless of the number of layers, basically computes a linear combination of the input features). There exist an endless number of non-linear activation functions, but one typically uses commonly applied (well-known) transformations that showed good results (Sharma and Sharma, 2017); there have also been approaches to making the model learn or adapt these functions (Agostinelli et al., 2014; Qian et al., 2018).

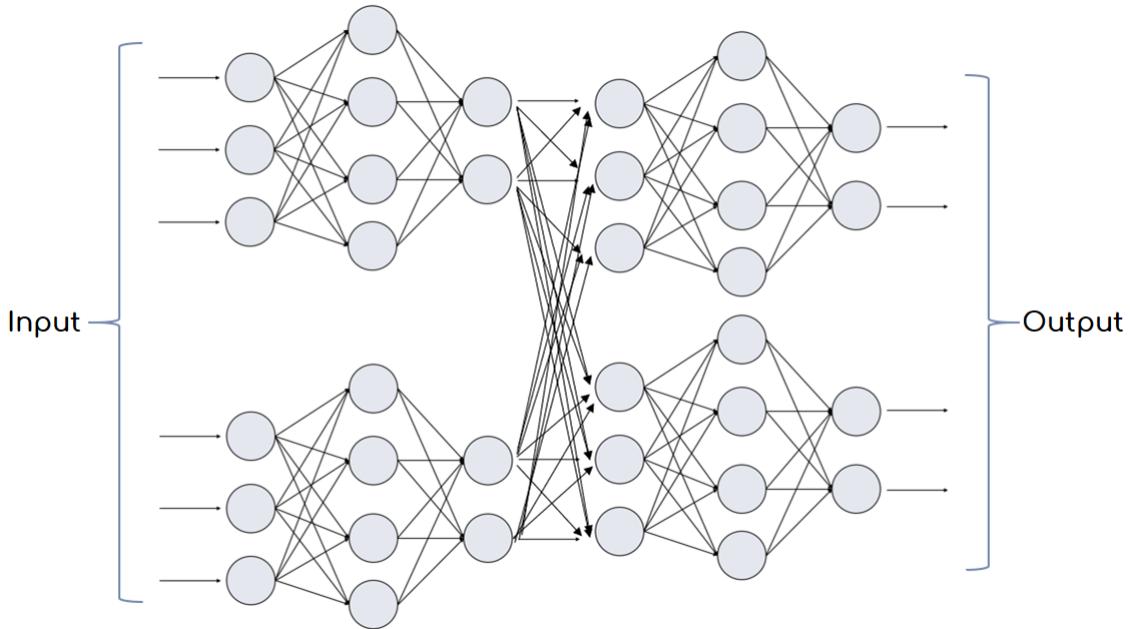


FIGURE 2.8: Exemplary modular combination of NNs into one larger network with eight input neurons and four output neurons.

In general<sup>2</sup>, during forward propagation, every neuron within an NN performs a *forward pass* by calculating a weighted sum of its inputs, incorporating a bias term, and then applying an activation function to generate an output. Together, these collective forward passes contribute to the overall forward (layer) propagation process. This process occurs layer by layer up to the final layer. Mathematically, for a neuron indexed by  $j$ , with input vector  $\vec{x}^{[j]}$ , weight vector  $\vec{w}^{[j]}$ , bias term  $b^{[j]}$ , and activation function  $\phi$ , its output  $y^{[j]}$  is given by:

$$y^{[j]} = \phi \left( b^{[j]} + \sum_{i=1}^{d^{[j]}} w_i^{[j]} x_i^{[j]} \right) \quad (2.7)$$

where each neuron has its own (single) bias; and the input and weight vectors both have the same dimension  $d^{[j]} \triangleq \dim(\vec{x}^{[j]}) = \dim(\vec{w}^{[j]})$ .

### Take-Away

**Neural Networks (NNs) are mathematical systems that draw inspiration from the structure and functioning of the human brain.** These systems comprise interconnected nodes called neurons, which are similar to the neurons in the brain that are connected by synapses. The purpose of NNs is to identify patterns and solve problems by processing information through these interconnected nodes. They take an input, transmit it through multiple layers of neurons, and generate an output. Deep Learning involves NNs with multiple hidden layers, which enables them to recognize complex patterns. NNs utilize activation functions that introduce non-linearities, essential for their ability to model complex relationships in data. Ultimately, NNs serve as powerful tools for tasks like classification, regression, image and speech recognition, natural language processing, and various other AI applications.

### Keywords

Neural Networks • Neurons • Deep Learning • Activation Functions • AI

## 2.8 Transformers

The *Transformer* model is considered a "revolutionary" architecture in the field of AI and especially in the area of natural language processing (NLP) (Lin et al., 2022; Ghoghoj and Ghodsi, 2020). The milestone was set by the paper - *Attention is All You Need* (Vaswani et al., 2017) published by Google Brain and Google Research members. This significantly outperformed existing state-of-the-art models; not only training faster but also achieving better final evaluation results. Transformer models have gained massive popularity among AI practitioners in industry and academia. Since then, many variations have emerged - e.g *Linformer* (Wang et al., 2020a), *Performer* (Choromanski et al., 2020), *Longformer* (Beltagy, Peters, and Cohan, 2020), *Reformer* (Kitaev, Kaiser, and Levskaya, 2020), etc. - and are referred to as "X-formers" (Tay et al., 2020). The application areas of transformers vary widely but are commonly

<sup>2</sup>In more complex NN architectures, the forward equation can be modified to fit specific structures, especially those with different layer types like convolutional and recurrent layers. For instance, Convolutional Neural Networks create feature maps by convolving filters; Recurrent Neural Networks update hidden states based on previous states.

applied to in NLP and computer vision tasks (Wolf et al., 2020a; Khan et al., 2021c). Figure 2.9 below shows an illustration from Vaswani et al., 2017 depicting the classic transformer model architecture.

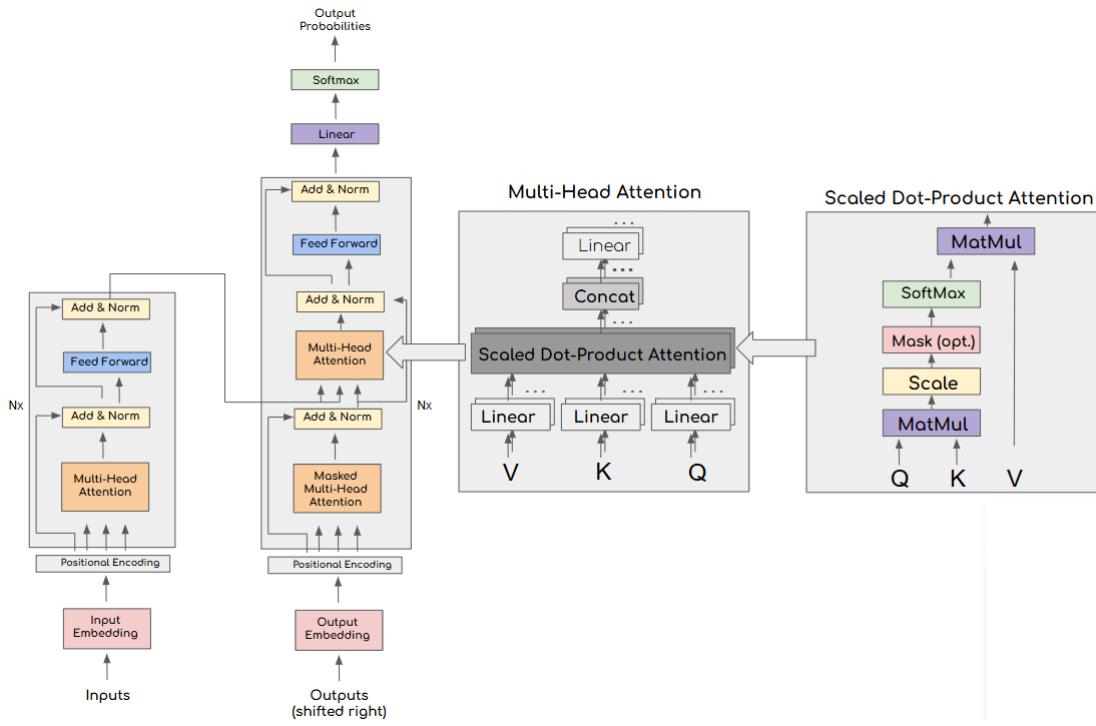


FIGURE 2.9: Transformer model architecture with multi-head attention and scaled dot-product [Architecture from Vaswani et al., 2017].

Transformers are a crucial component of Large Language Models (LLMs), which can be thought of as AI systems that can generate responses to user prompts that are similar to or resemble human-like responses. They are trained on massive text data sets using advanced DL techniques. LLMs and the attention mechanism used by them have completely revolutionized the field of NLP and are used in many applications such as text summarization, information extraction, question answering, text classification, conversation, code generation, reasoning, etc. During inference, LLMs are guided by a (user-defined) prompt (Saravia, 2022). In-context learning allows for temporary learning from these prompts. Prompt engineering is crucial in the use of LLMs. It involves selecting words, phrases, symbols, and formats to direct the model's output. Effective prompt design significantly impacts response length and detail, requiring creativity and precision.

Therefore, a Transformer is a specific type of DNN architecture, while LLMs represent scaled-up implementations of that architecture in size, parameters, and training data. The exact scaling factor from the original Transformer architecture to current LLMs, however, varies significantly depending on the specific LLM; but typically represents a very substantial increase (by several orders of magnitude).

There exist multiple different variants of LLMs (Saravia, 2022). Some of the most renowned transformer models and powerful LLMs that stand out as significant milestones in the field are shown in Table 2.1 below:

LLMs require complex architectures, pre-training, and innovative approaches to improve natural language understanding tasks. A brief summary outlining the fundamental key components and basis for LLMs derived from literature are:

TABLE 2.1: Top Large Language Models

Year (Month)	Model	Year (Month)	Model
2018 (June)	GPT	2020 (May)	GPT-3
2018 (Oct)	BERT	2020 (Jul)	BART
2019 (July)	RoBERTa	2022 (Mar)	InstructGPT
2019 (Nov)	GPT-2	2022 (Apr)	PaLM
2019 (Oct)	T5	2022 (Nov)	BLOOM
2019 (Jun)	XLNet	2022 (Nov)	ChatGPT
2019 (Sep)	ALBERT	2023 (Feb)	LLaMA
2019 (Sep)	CTRL	2023 (Mar)	GPT-4
...	...	2023 (Mar)	PanGu- $\Sigma$
...	...	...	...

#### - Transformers' Core Components:

1. **Self-Attention Mechanism:** Allows to efficiently capture word relationships; surpassing traditional networks in understanding long-range dependencies (Vaswani et al., 2017; Shaw, Uszkoreit, and Vaswani, 2018).
2. **Multi-Head Attention:** Improves focus on different aspects within input sequences by employing multiple attention weight sets (Vaswani et al., 2017; Cordonnier, Loukas, and Jaggi, 2020; Niu, Zhong, and Yu, 2021; Ma et al., 2019).
3. **Positional Encoding:** Provides sequential information by embedding the position of tokens within a sequence, helping the model differentiate between different positions (although not all models utilize positional encoding; and various versions exist). This facilitates learning and processing of long-term sequential relationships and dependencies (Dufter, Schmitt, and Schütze, 2022; Ke, He, and Liu, 2020; Chu et al., 2021; Lin et al., 2022).
4. **Feedforward Neural Networks:** Processes and transforms information obtained from the self-attention layers within the transformer architecture (Schmidhuber, 2015).

#### - Foundations for Large Models:

1. **Scale:** Larger models with a greater number of parameters to capture complex patterns; consisting of multiple self-attention and feedforward layers (Villalobos et al., 2022).
2. **Pre-training on Vast Text Corpora:** Models are prepared through tasks such as masked language modeling or next word prediction, which significantly improve contextual language comprehension (Kalyan, Rajasekharan, and Sangeetha, 2021; Edwards et al., 2020).
3. **Efficient Training Techniques:** Cutting-edge approaches such as gradient checkpointing and parallel computing are employed to minimize memory consumption and accelerate training. (Shen et al., 2023; Xiao et al., 2023).
4. **Transfer Learning:** Better downstream task performance and domain adaptation with less labeled data can be done by fine-tuning on (smaller) data sets of interest (Zhuang et al., 2020; Alyafeai, AlShaibani, and Ahmad, 2020).

**- Challenges and Progress:**

1. **Computational Demands:** High computational resources limit accessibility for researchers and organizations (Thompson et al., 2020; Hadi et al., 2023; Rillig et al., 2023).
2. **Ethical Considerations:** Concerns arise regarding biases, AI misuse, and environmental impact; necessitating responsible practices in development and deployment (Khan et al., 2021a; Weidinger et al., 2021; Head et al., 2023).
3. **Continual Advancements:** Ongoing research aims to improve efficiency, interpretability, and accessibility through techniques like sparse attention mechanisms and model compression (Shao et al., 2021). Furthermore, while the range of LLM variations in the technological core of AI is decreasing, the complexity of these technologies is increasing. This suggests that innovations within the essential AI technologies are becoming more distinct from each other despite being rooted in the same core. (Leusin, Jindra, and Hain, 2021).

**Take-Away**

The Transformer is an advanced technology based on Google's "Attention is All You Need" approach. It has revolutionized natural language processing (NLP) by enhancing speed and accuracy. The Transformer has been upgraded with different iterations like Linformer, Performer, and other "X-formers," for different applications. Transformers are the cornerstone of large language models (LLMs) and use vast amounts of text data to perform tasks like summarization and question-answering. They rely on self-attention, multi-head attention, positional encoding, and neural networks. LLMs are scalable and pre-train on huge corpora, making use of transfer learning and efficient techniques. However, there are challenges, like high computational demands, ethical concerns, and ongoing advancements focusing on efficiency and interpretability that make it different from other AI technologies.

**Keywords**

Large Language Models • Transformer • Attention is All You Need • Natural Language Processing • Foundations • Challenges.

## 2.9 Clustering & Kleinberg's Theorem

The concept of clustering is straightforward: given a diverse set of elements, group them together using "some sort" of similarity measure. These elements are commonly represented as a set of points in a (often high-dimensional) vector space. In the context of DNN, an intermediate latent (lower-dimensional) representation vector is frequently extracted and used for clustering, rather than original (high-dimensional) raw features. Ideally, points in the same cluster are close together and points in different clusters are far apart, i.e. similar points should be close together, while dissimilar points far apart. The term "similar" is hereby defined vaguely since it is often not clear or easy to define or specify what "similar" means; particularly in unsupervised settings. Therefore, the distances between the points are commonly used to represent or indicate similarities; the closer the points are, the more similar (hopefully) they are.

The idea of clustering may appear simple and straightforward at first, but building a general framework in practice is quite challenging, and various clustering methodologies and algorithms have been developed throughout the years (Rai and Singh, 2010; Xu and Tian, 2015; Berkhin, 2006; Xu and Wunsch, 2005). Indeed, the authors of "An Impossibility Theorem for Clustering" (Kleinberg, 2002) even presented an example of three simple properties for which no clustering function could satisfy all three at the same time, exposing inevitable trade-offs. Kleinberg's impossibility theorem outlines three basic, intuitive, and plausible key properties of a desired clustering function, and establishes that no clustering function can simultaneously fulfill all three properties:

1. **Scale-Invariance:** The clustering should remain unchanged when the data is equally stretched in all directions.
2. **Consistency:** If distances between clusters increase or within clusters decrease due to data stretching, the clustering shouldn't alter.
3. **Richness:** The clustering function should have the capability to generate any arbitrary partition of data points.

Follow-up work (Ben-David and Ackerman, 2008) proposed considering "clustering quality measurements" (CQM) as the object to be axiomatized rather than "clustering functions" and proposed a revised set of criteria (axioms) for such measures. That is, they argue that, even though no clustering function can exist that satisfies all of Kleinberg's properties, there can be a **CQM** that satisfies a set of axioms and evaluates the quality of clustering. The authors show that the *clustering-quality* framework is richer and more flexible than *clustering functions* because it allows the postulation of axioms that capture the features expressed by Kleinberg's axioms without producing a contradiction. They propose the CQM as a function that, given a data set and its clustering, determines how "good" the grouping is. Evaluation of clustering methods is thus important, due to the difficulty of developing a unified clustering framework that is independent of any specific algorithm, objective function, or generative model.

### Take-Away

Clustering groups elements by similarity in high-dimensional spaces, with the evaluation of methods crucial due to inherent trade-offs outlined by Kleinberg's theorem and the necessity for adaptable frameworks like clustering-quality measures.

### Keywords

Clustering • Similarity measures • Evaluation, Kleinberg's Theorem

## 2.10 Model Evaluation

The purpose of *model evaluation* is to assess the "effectiveness and quality" of a model. The quality evaluation metric or criterion does *not* necessarily have to correspond to the training loss function; in fact, it typically does not. E.g., not only may it not be differentiable, but its purpose may also be to consider the performance of a model

under a specific (e.g. even different) objective criterion. In particular, even if a training loss function is perfectly optimized (e.g. by a loss error of zero), this does not, by any means, guarantee that this respective loss function actually properly reflects the underlying problem task (and complexity) in the first place. However, directly optimizing for the real or intended target objective (i.e. loss) is often infeasible, too complex, or presents other issues and challenges; so a different (proxy) loss is used instead. The loss may also just be a heuristic (the true target objective is unknown).

It is important to be aware of the difference between both concepts. E.g., if we consider metrics like accuracy, precision, recall, or F1-scores, such metrics rely on binary predictions ("is" or "is not" of category X), whereas the loss function relies on a numeric output value. Here, we are free to choose any loss objective (e.g. F1, accuracy, etc.) for a given loss function (e.g. cross-entropy, logistic-regression, etc.).

### 2.10.1 Generalization & Overfitting

Model generalization refers to how well it performs on new, different (test) data. Overfitting is when a model *fails* to generalize and learn patterns for new data. These are fundamental concepts, very well-known, and extensively reviewed and explored in literature (Ying, 2019; Hawkins, 2004; Srivastava et al., 2014). Overfitting can happen due to a variety of reasons, such as (excessive) training noise, learning too specific, uncommon details, or when training on data that is not well representative of the overall real data distribution; but also due to many other reasons like insufficient training data, too complex model architectures with too many parameters, leaking target information into features (data leakage), highly correlated features, improper hyperparameter tuning, outliers in the training data, imbalanced data and labels, (excessive) feature engineering, etc. Techniques like regularization, cross-validation, and early stopping help mitigate overfitting (improve generalization). For details on overfitting and solutions, we refer back to the respective literature; and Section 2.2.7.

### 2.10.2 Clustering Evaluation

For assessing the cluster quality, there are two main approaches (Amigó et al., 2009): *intrinsic* and *extrinsic* methods. (i) Intrinsic evaluation examines the quality of clusters based on data or characteristics without ground truth labels. (ii) Extrinsic evaluation compares clusters against external benchmarks or ground truths (i.e. labels).

#### Intrinsic Clustering Evaluation

Intrinsic clustering evaluation metrics are ways to evaluate the effectiveness of clustering algorithms without relying on external information, such as labels. These metrics assess the quality of clusters formed within data and are thus called "intrinsic" metrics. Common metrics include the Silhouette Coefficient (cohesion within clusters and the separation between clusters), Davies-Bouldin Index (cluster compactness and separation), Calinski-Harabasz Index (cluster dispersion ratio), Dunn Index (cluster compactness and separation), Gap Statistic (optimal cluster number), etc. Each metric focuses on different aspects. Choosing the right metric depends on the data set and goals. For a deeper understanding, we encourage the reader to review the corresponding literature (Xu and Wunsch, 2005; Wegmann et al., 2021; Tomašev and Radovanović, 2016).

## Extrinsic Clustering Evaluation & Uncertainty

**Notice:** Our work and findings on  $\alpha$ Max-B-CUBED (presented in this thesis) have been publicly released on OpenReview-Venue (Guimerà Cuevas and Schmid, n.d.).

As already mentioned before (Section 2.2.4), Weak Supervision is a branch of ML in which the model is trained using noisy, incomplete, or inexact annotations instead of complete and accurate ones (Zhou, 2018). A model may be designed to deal with noise and uncertainty in annotations to make the best possible predictions based on the information given. Because producing high-quality labels is typically very costly, weak supervision is often used to generate additional cheaper, but lower-quality labeled data. "Inaccurate training data" contains defects or erroneous labels. The term "inexact data" refers to training data with imprecise labels, such as coarse categories or probabilistic labels. Inexact supervision can result in a model with lesser prediction confidence, but also in misleading conclusions during supervised evaluation of clustering results. Cluster quality evaluation with inexact labels refers to the process of evaluating the performance of a clustering method when the ground truth labels are not fully known or are too generic. "Incomplete training data" refers to data that lacks key information or characteristics.

Therefore, evaluation metrics and clustering algorithms that address or ensure cluster quality under inexact, inaccurate, or incomplete data or labels (i.e. uncertainty) are important here. Data clustering with partial supervision, where data is neither completely nor accurately labeled, was e.g. approached using a fuzzy clustering-based technique that uses available data knowledge to supervise the clustering process (Bouchachia and Pedrycz, 2006). The adjusted Rand Index (Rand, 1971) measures the similarity between the real and predicted (cluster) labels while adjusting for chance; related to accuracy. Here, "chance" refers to the possibility of achieving a specific outcome by random. This involves considering that the agreement between the true and predicted labels might occur by (pure) coincidence; especially for clusterings that do not appropriately group data points. Normalized Mutual Information (Press et al., 2007) assesses the mutual information between the real and predicted cluster labels, normalized by the entropy of both; where entropy can be regarded as a measure of uncertainty. The Fowlkes-Mallows index (FMI) (Fowlkes and Mallows, 1983) computes the geometric mean of precision and recall between true and generated clusters. FMI tries to measure uncertainty by calculating precision and recall over both the number of correct and incorrect predictions.

Determining the quality of clustering results is a non-trivial task and the interpretation of clusters is commonly hindered by the lack of objective criteria (Rand, 1971). In a famous work on extrinsic clustering evaluation (Amigó et al., 2009), the authors propose formal constraints on cluster quality evaluation metrics. They emphasize that metrics should be intuitive, clarify limitations, be formally provable, and allow to distinguish between different metric families based on their mathematical foundations. They present four constraints on quality measurements

Amigó et al., 2009 demonstrate that out of many commonly used metrics (e.g. set matching metrics, counting pairs metrics, entropy metrics, edit-distance based metrics, etc.), only the **B-CUBED** ( $B^3$ ) metric (Bagga and Baldwin, 1998) satisfies all four constraints, while the others do not.

$B^3$ , however, assumes that labels are accurate and represent a "ground truth". This leads to issues on inexact or coarse labels (i.e. label uncertainty) (Chapter 3.3). Uncertainty in labels can harm the accuracy and quality of the evaluations as we cannot accurately assign data points to their true (real) labels; i.e. if the labels are not clearly defined or well-known, any evaluation depending on it will be uncertain.

**Take-Away**

Weak supervision involves training a model using incomplete, inaccurate, or inexact labels. When the annotations are inexact, it can lead to a model with lower prediction confidence. Model evaluation goes beyond optimizing the loss function and involves assessing the "effectiveness" of a model using different metrics, such as accuracy, precision, and recall. For clustering, there exist various (supervised) metrics to evaluate the cluster quality. However, among many metrics, only  $B^3$  satisfies four fundamental formal constraints simultaneously.  $B^3$  assumes accurate labels as a ground truth, which can cause problems when dealing with coarse labels and uncertainty.

**Keywords**

Weak Supervision • Model Evaluation • Clustering Quality •  $B^3$  Metric



## Chapter 3

# Motivation

### 3.1 Neural Networks Based on Complex Numbers with Weights Constrained along the Unit Circle

**Notice:** Our work and findings on complex-valued NNs (presented in this thesis) have been published in Guimerà Cuevas, Phan, and Schmid, 2023. It was inspired and builds upon our previous research (Guimerà Cuevas and Phan, 2021), but differs significantly as it has been substantially improved and modified (both the complex architecture and its mathematical framework); thus, constitutes novel contributions. In particular, this novel contribution has been accepted for publication, after peer review. The Version of Record is available online at: [https://link.springer.com/chapter/10.1007/978-3-031-33374-3\\_28](https://link.springer.com/chapter/10.1007/978-3-031-33374-3_28). Use of this Version is subject to the publisher's Manuscript terms of use. Therefore, please refer to Guimerà Cuevas, Phan, and Schmid, 2023 for specific details.

Traditional real-valued neural networks (NNs) can diminish neural inputs by assigning weights of zero or overpower other inputs through extremely large weights. However, large weights are undesirable as they can destabilize the network and cause exploding gradients, necessitating effective regularization. This thesis proposes a new architecture for feed-forward and convolutional layers that restricts weights to the unit circle, ensuring that neural connections cannot be entirely removed - which guarantees that no incoming information is lost due to inactive neurons - or weakened by the weights. The neural network's decision boundaries are redefined by representing weights as phase rotation angles and inputs as amplitude modulations, with the trainable weights consistently maintained within a fixed range. Our method can be seamlessly incorporated into existing layers without altering the original network structure. The classification performance was validated on standard computer vision datasets using ShuffleNetv2, ResNet18, and GoogLeNet at high learning rates.

#### Take-Away

Enforcing weight constraints within the unit circle in the neural network architecture can prevent or counteract input suppression or information loss.

#### Keywords

Neural Architecture • Complex Numbers • Unit Circle • Weight Constraints

## 3.2 Deep Learning for Hierarchical Databases with Recursive One-to-Many Relations

To generate or learn HDB embeddings, the first step is to devise an efficient and effective encoding method for the raw HDBs, coupled with a model architecture capable of processing the respective encoding type effectively.

Standard deep learning (DL) methods, however, face a major challenge when dealing with hierarchical databases (or graphs) that contain recursive one-to-many relationships (HDBs). HDBs are typically heterogeneous with complex topological inter-relationships, which traditional neural architectures are unable to process directly due to their requirement for a fixed-size linear vector representation of the input data. This architectural limitation renders them incapable of supporting non-linear graph structures, which is essential for processing HDBs. The term "non-linear" here refers to the inadequacy of a simple linear encoding of raw HDBs to capture heterogeneous features, structures, and topology.

Consequently, processing hierarchical graph-structured data (i.e. HDBs) requires specific techniques that can handle their complex characteristics, such as variable-sized inputs, local and relational information, graph-specific model (feed-forward) operations, permutation invariance, and preservation of topological and structural information. Traditional DL methods fall short here of adequately handling these complexities; (brute-force) linearly encoding graphs and forwarding them naively through a DNN is, overall, not a good solution. To process hierarchical graph-structured data most effectively and directly, targeted solutions are necessary.

Homogeneous Graph Neural Networks (GNNs) are also unsuitable for HDBs since they are designed to operate on graphs with identical node and edge features. Although specialized heterogeneous GNNs are capable of representing and processing heterogeneous graphs, they may not be optimally designed for HDBs because they do not directly leverage the graph's hierarchy and its loop-free relational structure. Therefore, we propose a new DL architecture designed specifically for HDBs. Unlike heterogeneous GNNs, our method combines and aggregates topological and feature information by using a uni-directional bottom-up forward propagation approach in which higher-level representations rely solely on lower-level representations. We validate our approach exemplary on synthetic and real-world hierarchical Rebrickable® data in supervised and unsupervised tasks, showing its effectiveness and outperformance of naive brute-force transformer-based and GNN approaches.

### Take-Away

Standard Deep Learning struggles with complex hierarchical databases due to their heterogeneity, recursive nature, and variable input size.

### Keywords

Hierarchical databases • Graph Neural Networks • Recursive relationships

## 3.3 Addressing Uncertainty and Completeness in the $B^3$ Cluster Evaluation Metric

**Notice:** Our work and findings on  $\alpha$ Max-B-CUBED (presented in this thesis) have been publicly released on OpenReview-Venue (Guimerà Cuevas and Schmid, n.d.).

Assessing the quality of clustering results is a crucial and challenging task. The B-CUBED ( $B^3$ ) precision and recall evaluation metric has gained popularity due to its ability to meet four formal constraints: homogeneity, completeness, rag bag, and size vs. quantity. However, the ‘completeness’ constraint, which demands that items of the same category be grouped in the same cluster, can pose problems for finer clustering algorithms that identify sub-clusters within clusters. This issue is particularly pronounced when the available labels are imprecise and coarse, resulting in uncertain and fuzzy cluster evaluations.

To address this, we propose a modified evaluation metric:  $\alpha$ Max- $B^3$ . Our approach accounts for completeness and uncertainty in subgroup evaluation by reorganizing clusters into super-sets based on the most prevalent label and evaluating them alongside the original clusters using a modified weighted  $B^3$  metric. The extent of uncertainty, given by  $\alpha$ , can be either explicitly specified or automatically estimated.

#### Take-Away

A modified evaluation metric called  $\alpha$ Max- $B^3$  is proposed to address the issue of completeness and uncertainty in subgroup evaluation in the  $B^3$  clustering algorithm.  $\alpha$ Max- $B^3$  reorganizes clusters into super-sets based on the most prevalent label and evaluates them alongside the original clusters using a modified weighted  $B^3$  metric. The extent of uncertainty, given by  $\alpha$ , can be specified explicitly or automatically estimated.

#### Keywords

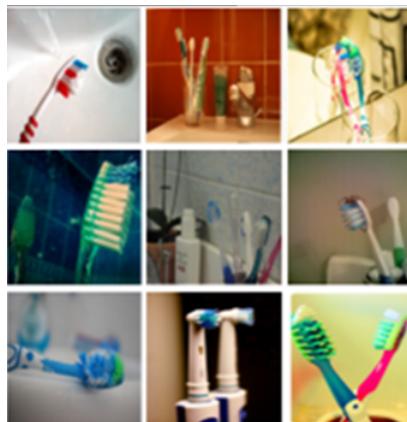
$B^3$  Clustering • Completeness • Uncertainty • Modified Evaluation Metric

## 3.4 Automated Textual Description Generation of Clusters

Generating accurate and concise descriptions for image clusters involves a complex intersection of computer vision and natural language processing. This thesis underscores that basic statistical and geometrical manifold aggregation methods on neighboring cluster instances can already produce surprisingly effective results without additional training. By simply leveraging the inherent manifold information within a generative model, different representative textual cluster descriptions can be generated and sampled. When combined with a population-based decoding strategy, these descriptions can further be substantially improved. *In other words:* already by using simple mathematical techniques on groups of nearby manifold points, we can effectively generate descriptions for clusters (without extra training); we can further improve the description quality using additional (more special, targeted) strategies.

We explore various pooling and decoding methods and find that even very simple (naive) aggregations like the average cluster embedding vector can yield concise and representative cluster descriptions. We observe that the effectiveness of the text decoding depends on the pooling approach chosen. Moreover, we introduce and compare two contrastive Language-Image-based pooling methods: a gradient-guided pooling optimization for a unified manifold, and a contrastive weighted averaging approach. Our experiments are evaluated on the COCO-2017 (Lin et al., 2014) and TextCaps (Krause et al., 2017; Sidorov et al., 2020) data sets.

Finding "any" clusters during clustering is always possible. However, the ultimate question is: what do these clusters represent, and how useful and relevant are the clusters? This is where cluster explanation or interpretation (XAI) comes into play. We want short, straightforward descriptions of arbitrary but fixed clusters. That is, once we've identified potential clusters, it would be really helpful if we could extract some (textual) explanation for these clusters (Figure 3.1). The benefits of adopting natural language for explanation are that (1) it is simple to comprehend and interpret, (2) it allows us to highlight keywords in the sentence that are crucial for a specific cluster assignment, and (3) it helps us to determine semantic similarities across different clusters.



A couple of toothbrushes on a bathroom countertop.

FIGURE 3.1: Using language to explain clusters: A text comment is created for a cluster of images that serves as a common description of all instances; [Figures from TextCaps data set (Sidorov et al., 2020)].

#### Take-Away

Computer vision and NLP can be combined to create textual descriptions for image clusters. Textual explanations for clusters aid comprehension and can highlight essential keywords, enhancing cluster interpretability.

#### Keywords

Clustering • Interpretability • Computer Vision • NLP • Textual Explanation

## 3.5 Normalization of Heterogeneous Feature Distributions

**Please note:** This contribution has been accepted for publication, after peer review in Guimerà Cuevas and Schmid, 2024b. The Version of Record is available online.

Heterogeneous feature distributions occur when the numeric values of the features within a data set are distributed differently, indicating dissimilarities and variations in their characteristics. Dealing with heterogeneous feature distributions in data analysis or ML tasks requires employing methods like normalization, feature scaling (FS), or other specialized algorithms to help align the distributions and enhance the performance and reliability of the analysis or ML tasks.

ML models rely on structured, machine-readable data representations. Ensuring numerical values are appropriately scaled is therefore crucial for optimal processing.

Improper or poor FS can lead to skewed or biased models with imbalanced weights assigned to features, causing inefficient or ineffective learning; making it difficult for the model to converge effectively to an optimal solution. Traditional methods of FS also fall short in handling outliers within data sets, which can disrupt statistical assumptions and further diminish model accuracy and performance.

Non-linear Tanh-Estimators (TE) have been shown to provide a robust approach to feature normalization (i.e. FS). However, they apply a global fixed scaling factor to all features, raising concerns regarding its applicability to diverse, heterogeneous feature value distributions. In this thesis, we propose an improved TE refinement that utilizes the Wasserstein distance to estimate the optimal scaling factor of each feature distribution; making the individual normalized feature output distributions most similar to a given (ideal) target distribution, such as a standard Gaussian.

Our results show that such a feature-adaptive normalization approach can clearly outperform the currently established TE method in the literature in terms of convergence speed by allowing for better initial training starts by reducing or eliminating the need to re-adjust model weights during early training phases due to inadequately scaled features. Therefore, in this thesis, we aim to highlight, explore, and present an analytical perspective on the (bad) practice of utilizing a single constant global spread value in the context of FS for numeric data representations; and propose a solution. We empirically evaluate our approach on common toy computer vision data sets, synthetic data, and a real-world tabular data set.

#### Take-Away

Utilizing Tanh-Estimators with Wasserstein distance-optimized spread values for adaptive feature scaling enhances convergence speed and robustness.

#### Keywords

Feature Scaling • Outliers • Tanh-Estimators • Adaptive Normalization

## 3.6 Out-of-Distribution Detection

**Please note:** Our work and findings on Calibration Misalignment as a Post-hoc Approach for Out-of-Distribution Detection in Deep Neural Networks (presented in this thesis) have been accepted for publication in conference proceedings (Guimerà Cuevas and Schmid, 2024a).

Out-of-distribution (OOD) detection plays a crucial role in the broader field of outlier detection, as it focuses on distinguishing between instances that conform to the distribution of the training set and those that are anomalous. The primary objective of this work is to propose a novel post-hoc method for identifying OOD examples in DNNs. In this thesis, we present a methodology that leverages the misalignment of multi-class output calibration to detect OOD examples effectively. To quantify the disparity in output probability calibration for individual classes, we adopt a one-vs-rest approach that considers the normalized total probability mass across all class calibrations. Furthermore, we consider the discrepancy in probability mass between the combined probabilities of class pairs and the expected sum of their individual calibrated probabilities. This magnitude of probability mass misalignment serves as a valuable indicator for identifying OOD examples. To validate the effectiveness of our approach, we conducted experiments that demonstrate how

the proposed discrepancy and probability misalignment successfully differentiates between OOD examples and in-distribution examples. Notably, our method detects anomalous behavior when presented with outliers, showcasing its ability to identify OOD examples. Additionally, we compared our prediction-based OOD approach with common embedding-based anomaly detection methods on widely used computer vision data sets. Encouragingly, our method outperformed these traditional approaches, further highlighting its potential and applicability in real-world scenarios. OOD detection is key to ensuring the reliability and robustness of ML models.

#### Take-Away

Leveraging discrepancies and misalignments in output probability calibrations can help detect out-of-distribution examples within DNNs.

#### Keywords

Out-of-distribution • Probability Calibration • Anomaly Detection • DNN

## 3.7 Manifold Data Representation and Clustering

Data exploration relies on assumptions like *smoothness & continuity* (points close in the input space are likely to have the same label/output), and small input changes yield gradual output changes), *manifold presence* (data lies on low-dimensional manifolds, but is embedded in high-dimensions), and *clustering* (data form distinct clusters; points in the same cluster are more likely to share labels/outputs), to uncover patterns or structures within the data. Based on these assumptions, we propose a hypothesis: by manipulating the latent representations of data and creating an artificial manifold space, it may be possible to discover subgroups within (labeled) data. To achieve this, we aim to minimize manifold approximation errors for similar labeled points and maximize these errors for dissimilar labeled points. This is because instances on the same manifold are likely to have distinct characteristics that form separate clusters. Therefore, nearby points that can be accurately approximated based on the local shape of the manifold are more likely to belong to the same subgroup; i.e. compared to distant points or those with poor approximations.

#### Take-Away

Analyzing latent manifolds may help in identifying subgroups.

#### Keywords

Data Assumptions • Manifold Approximation • Latent Spaces • Subgroups

## 3.8 Representative Data Stream Sampling for Trajectories

Applying ML to analyze patterns within data stream trajectories poses a significant challenge due to their continuous influx of new data, often arriving at varying rates or intervals. This variability can differ not only across different streams or input signals but also within the same stream. Additionally, employing DL, outlier detection,

real-time classification, and clustering within this context faces further challenges when dealing with the dynamic, evolving, and intermittent nature of these streams or trajectories (e.g. streams terminating at different time-stamps or temporarily halting for an unknown length but still continuously gathering data from the most recent state). Managing the complete history of multiple data streams concurrently becomes non-trivial and difficult due to memory limitations, requiring an algorithmic or strategic approach to determine which trajectory records should be preserved and which should be discarded. Discarding a record deletes it forever from memory; and is deemed unrecoverable.

Addressing this, we propose a method centered around reservoir sampling, aiming to uphold a representative trajectory at every time step. To achieve this, we maintain a representative sample sequence within a designated quantile window. However, devising a solution is challenging as the size of the quantile window is not fixed; it dynamically expands with the length of the data stream. The key challenge lies in sampling "representative" data stream trajectories while being unable to store the entire sequence in memory. Consequently, with each new incoming data element, the decision must be made whether to include it or permanently discard it. Focusing on different quantiles provides a means to analyze specific segments of the trajectory, such as newer samples, allowing for targeted analysis within this complex data environment.

A quantile window provides distinct advantages over fixed sliding windows or basic reservoir sampling in certain scenarios due to its adaptability and ability to capture varying (recent) data distributions more effectively: (1) Quantile windows dynamically adjust their size according to the stream's length, maintaining a consistent proportion of data points rather than a fixed count. This adaptiveness may prove particularly advantageous when handling fluctuating data arrival rates and different input densities. In contrast to fixed sliding windows, which retain a set number of recent data points, representative sampling accommodates such varying data densities, ensuring a more representative sample. Also, in situations with skewed or irregular data distributions, fixed sliding windows might only capture a limited subset of the data and thus patterns. However, sampling windows can here efficiently represent the distribution without needing to store the entire data sequence or know the distribution in advance. (2) Moreover, focusing on different quantiles, such as newer or older sample intervals, quantile windows can facilitate targeted exploration of specific segments of interest within the data. This may yield better insights than using a fixed sliding window, especially when distinct changes or trends occur in various parts of the data stream.

Ultimately, the choice between different windowing or sampling approaches depends on the specific requirements, characteristics, or parts of the data stream under analysis and the intended goal. For example, some analyses might prioritize real-time processing and adaptability, while others focus on accurately representing the general distribution and identifying non-specific, overall trends in the data.

**Take-Away**

Examining data stream trajectories through Machine Learning presents challenges owing to the continuous influx of new incoming data and limitations in memory capacity. In this context, conventional reservoir sampling can uphold a representative sample for every timestamp. Quantile windows, depending on the scenario, can provide advantages over fixed sliding windows or traditional reservoir sampling due to their flexibility in adjusting quantile window sizes to accommodate various data lengths. In general, determining whether to use windowing or sampling methods depends on the precise needs and objectives of the analysis of the data stream.

**Keywords**

Data Stream Trajectories • Reservoir Sampling • Quantile Window

## Chapter 4

# Introduction

Over the last few decades, the volume of data has grown exponentially and become increasingly volatile (see Chapter 2.1; six-Vs). Many tasks within the industry have become costly, time-consuming, and prone to errors; making them increasingly difficult to manage manually. Without automation, these tasks can become tedious and often impossible to track efficiently. In the previous chapter, we motivated the importance and necessity of Artificial Intelligence (AI), explained why common rule-based approaches cannot keep up with modern complex problems (such as natural language), and discussed some of the main concepts, approaches, and domains within the field of AI.

In this chapter, we introduce the central research topic forming the backbone of this doctoral thesis. We highlight major challenges and specifically focus on topics addressing the essential demand for Machine Learning (ML) methods (i.e. AI systems, strategies, or algorithms) for automated error-pattern detection; which is particularly relevant in the industry. "Automated error-pattern detection" recognizes errors/patterns within systems or data; minimizing or eliminating the need for human intervention and oversight. Automated pattern detection techniques have many applications and use cases across various tasks and domains: they can be utilized to detect anomalies and recurring patterns, predict outcomes, classify and group new data instances, identify similar patterns or inputs, facilitate decision-making, conduct efficient analyses, and overall enable a wide array of other use cases, solutions, and downstream tasks.

**Disclaimer:** To maintain full compliance with imposed industrial privacy guidelines, any potentially sensitive information has been anonymized. As a result, the examples provided are chosen as purely illustrative and do not contain any real data or represent exemplary scenarios.

## 4.1 Framing the Context: Use Cases, Challenges, & Data

This section outlines the application context and data structure of the thesis for automated ML and pattern recognition; starting with an illustrative example for better understanding.

### 4.1.1 Framing the Context: Example Use Cases

#### Plant Growth Tracking in Greenhouses: Multifaceted Observation Records

Imagine a use case of observing plants growing in a greenhouse (see Figure 4.1). Each time a plant is watered, a record is generated and stored in a large database. These plant records are created for each watering, and a plant can have multiple

records depending on the frequency of its watering. Importantly, different plants may require different watering schedules; and thus such records can be produced at different rates. We will refer to these records as "observations". Some recorded features, e.g. the type of fertilizer used or the light intensity, might be irrelevant overall but are still recorded and included in the observation record just in case they could be useful somehow. Each observation can contain various types of data. Let us assume the primary categories are plant-, branch growth- and environmental sensor data (and so, we have multiple interconnected HDBs).

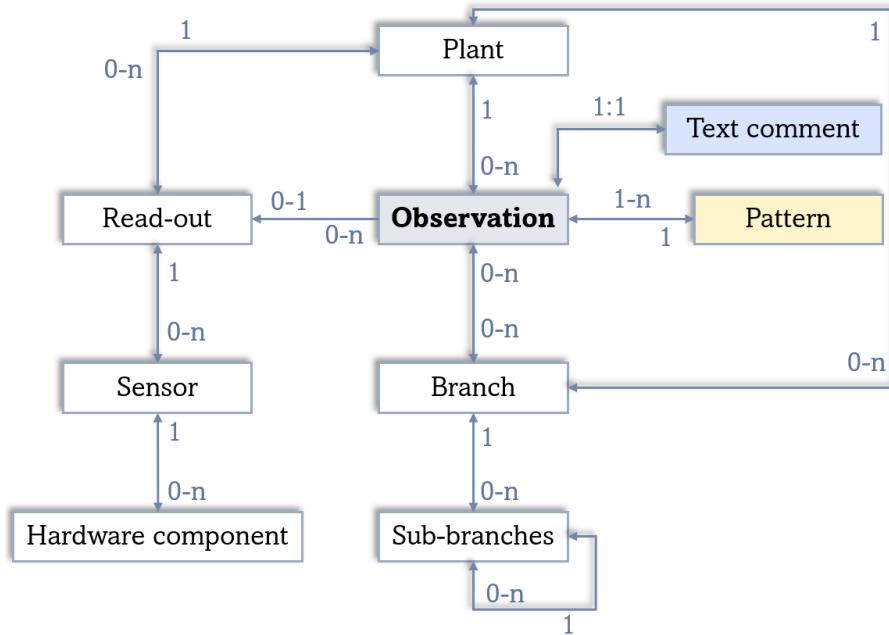


FIGURE 4.1: Visual representation highlighting the structure, dependencies, and relationships within a subset of sensor data. The data structure depicted is a hierarchical and recursive network of interconnected databases. Each database can link to additional sub-databases, all encapsulating a root database ("observation").

Plant data may store general information about the species of the plant. Environmental data consists of multiple sensor data captured as a snapshot from a moment of interest (e.g. the plant is checked for pests, or from the actual moment it shows signs of dehydration) via a read-out. Sensor data can consist of specific information about growth, leaf color, and pest infestation, and can be gathered by multiple sensors, but different plants may have different sensors and thus collect different types of data, potentially in different rates, quantities, and qualities. Additionally, one observation may record the growth of multiple branches, which themselves consist of several sub-branches, and recursively so on. Some observations might only record slight growth, whilst others might record significant growth. Data can, therefore, differ greatly. Environmental conditions, such as temperature and humidity, might also vary depending on the climate or time of day the observation was made, which can have different implications for each plant species (e.g. a succulent plant will require less water compared to other species and will show different hydration levels in the observation record). Plants do not always have comparable features and can differ in their observable characteristics, such as the development of flowers or fruit.

Furthermore, for each observation, the responsible employee has the freedom to

draft a custom report that can cover various aspects, such as noteworthy observations or potentially suspicious behavior. As these reports are subject to individual interpretation and can be influenced by personal mood, the quality and quantity of the information contained within can vary significantly among different employees.

Figure 4.2 below illustrates the data acquisition procedure graphically:



FIGURE 4.2: The process of data collection: Different plant species are examined by different employees who collect the data and transmit it to a centralized database. [Images created via DALL-E 3, 2023].

Moreover, these comments may be written in different languages, depending on the person, although the majority are (translated) into English. These comments do not follow any standardized format and can be considered "free-text.", i.e. service personnel have the freedom to rephrase the comments using their own words, resulting in variations in formality and vocabulary. There are multiple comment fields available to address various aspects of plant observation separately, but the service personnel might not necessarily follow these. This means that even though each comment field is designated for specific aspects of plant observation, personnel might e.g. copy-paste the same text into all fields to avoid leaving them empty. They may also write the wrong text in the wrong field or leave some fields empty altogether if they simply do not care. And occasionally, when in a bad mood, some might even deliberately write inaccurate or nonsensical information.

### Hierarchical Data Structures: Entities, Relationships, and Attributes

Consider a hierarchical data structure composed of primary entities containing subsidiary elements (Figure 4.3). An illustrative instance of such structures involves envisioning a country as a fundamental entity. Each country can be decomposed into different regions, wherein the number of regions may differ from one country to another. These regions further comprise diverse localities such as cities, villages, or towns, each potentially (but not necessarily) containing additional entities like schools, medical facilities, public transportation stations, etc.

For instance, a school, as one of these subsidiary entities, can be classified into different types including universities, primary schools, kindergartens, and others. Within a school, there exist multiple components like classes; which again further encompass different students or pupils. Delving deeper, each of these entities features a set of different characteristics of various types and sizes. I.e., a country may have characteristics like GDP, spoken languages, population size, neighboring countries,

and climate. Conversely, pupils may exhibit traits such as name, gender, academic performance metrics (such as average, best, and worst grade), number of friends, and nationality. Hence, not only is the structure of the linkage between these entities crucial, but also very importantly the specific values associated with the different attributes; thus both the structure itself and its accompanying features.

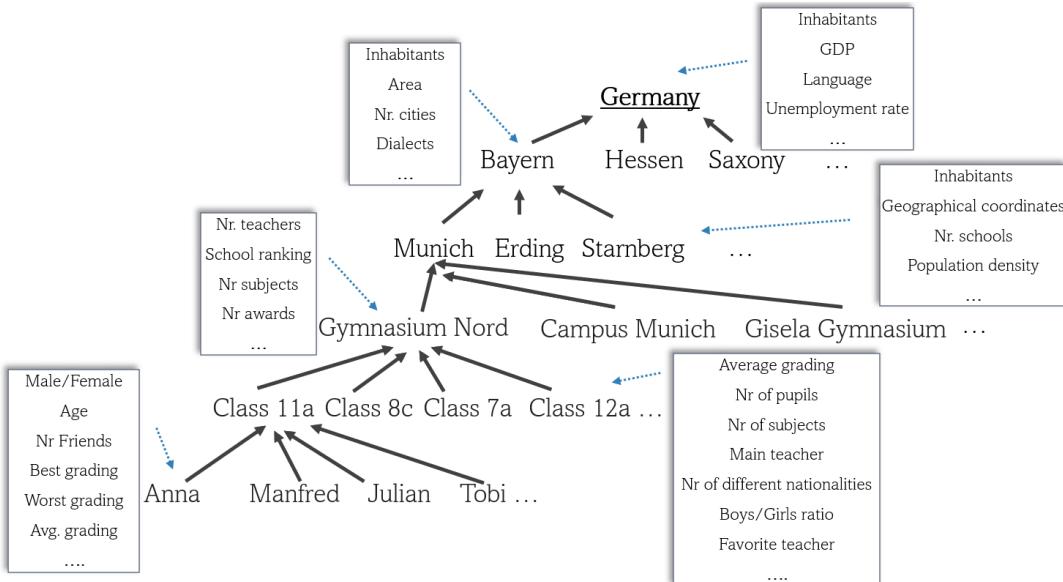


FIGURE 4.3: Hierarchical data structure composed of a primary root entity containing subsidiary elements.

#### 4.1.2 Framing the Context: Formal Definition

Consider a database consisting of multiple sub-databases connected in a hierarchical structure, creating a directed database graph network without any loops. Each database may encompass different features and feature types, and not all instances of a given database must include all features; but must be a subset of it. A compact high-level illustration depicting the hierarchical data dependency graph with recursive  $1:n$  connections is given in Figure 4.4 below:

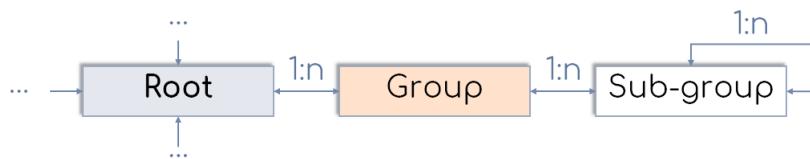


FIGURE 4.4: Abstract illustration depicting a hierarchical data dependency graph, visualized recursively. Each group (entity) is intricately connected to  $n$  subordinate entities, maintaining a strict hierarchical structure where connections ascend downwards (i.e., *no* upward or lateral movements within the same hierarchy level), resulting in a strict  $1:n$  hierarchical database. The hierarchical nature ensures a tree-like structure, inherently free of cycles.

This data structure (Figure 4.4) can be formally represented through the use and notation of Graph Theory. In this context, the hierarchical data structure can be represented as a special case of a Directed Acyclic Graph. Let  $G$  denote the graph representing the data structure, and let  $V$  and  $E$  represent the set of vertices (database entities) and edges (database relationships) in  $G$ , respectively. The graph  $G$  can then be characterized and defined by the tuple  $G := (V, E, \Omega)$ , where:

- $V = \{v_1, v_2, v_3, \dots\}$  is the set of vertices  $v$ , each representing an entity.
- $E = \{(v_i, v_j) \mid v_i, v_j \in V\}$  is the set of edges, with  $(v_i, v_j)$  indicating a directed edge from vertex  $v_i$  to vertex  $v_j$ .
- $\Omega_v = \{\omega_1, \omega_2, \omega_3, \dots\}$  is the set of feature attributes for an entity  $v \in V$ , where  $\omega$  denotes individual attributes (i.e. features).
- $\Omega = \{\Omega_{v_1}, \Omega_{v_2}, \Omega_{v_3}, \dots\}$  is the set of attribute sets for all entities  $V$ .

The edges in  $E$  adhere to a direct superiority relation denoted as  $\prec$ . If  $(v_i, v_j) \in E$ , it implies that vertex  $v_i$  is directly superior to vertex  $v_j$  within the hierarchy, and so:

$$(v_i, v_j) \in E \implies v_i \prec v_j \wedge \neg \exists v_k \in V : v_i \prec v_k \wedge v_k \prec v_j \quad (4.1)$$

The *acyclic* nature of the graph implies that no sequence of edges forms a cycle:

$$\neg \exists v_1, v_2, \dots, v_n \in V : (v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n), (v_n, v_1) \in E \quad (4.2)$$

This means we cannot start at a vertex, follow directed edges, and return again to that same vertex (which would be a cycle). This property is key for maintaining the *hierarchical* DB structure. In other words, the hierarchical data structure  $G$  is formally characterized as a set of vertices  $V$  (DB entities), interconnected by edges  $E$  without loops (DB relationships), and each entity  $v \in V$  has its set of attributes  $\Omega_v$ .

Given the set of all DB entities  $V$ , let  $\mathbb{D}$  be the set of all DBs. Each entity  $v \in V$  is associated, denoted by  $v \mapsto D$ , with (exactly) one DB  $D \in \mathbb{D}$ . The following holds:

$$\forall v \in V, \exists! D \in \mathbb{D} : v \mapsto D \quad \text{and} \quad \forall v \mapsto D : \Omega_v \subseteq \Omega_D \quad (4.3)$$

where  $\exists!$  is the uniqueness quantification ("there exists one and only one");  $\Omega_D$  is the feature set of  $D$ . In simpler terms, for every entity  $v$  in  $V$ , there exists a unique DB  $D$  in  $\mathbb{D}$ , such that  $v$  is associated with  $D$ , and the feature set of  $v$  is a subset of the feature set of  $D$ ; i.e., the set of features  $\Omega_v$  that an entity  $v$  can hold is always a subset of the feature set  $\Omega_D$  of its corresponding DB  $D$ . An entity is a *concrete* realization (i.e. instance) of its DB  $D$ ; but may not necessarily exhibit all its features. Different entities  $v \in V$  can, hence, have different feature attributes  $\Omega_v$ ; they may vary in both type and the number of features (count) across different DBs; within the same DB, only in count. Each  $v$  is linked to a single  $D$ , but each  $D$  can be linked to multiple  $v$ .

### 4.1.3 Composition of Multiple Hierarchical Data Sources

It is possible to integrate multiple independent HDB structures or data sources into a single, larger unified DB (Figure 4.5). This allows input data to include features from diverse aggregations of multiple different HDBs or sources, each with potentially varying levels of importance, relevance, features, meaning, quality, and size. Such sources can include not only other HDBs but also other feature input types like textual documents, images, etc. However, this integration poses further challenges,

such as longer processing times for intricate data sources (e.g. larger [nested] HDBs) and potential performance bottlenecks during training; if e.g. trained end-to-end.

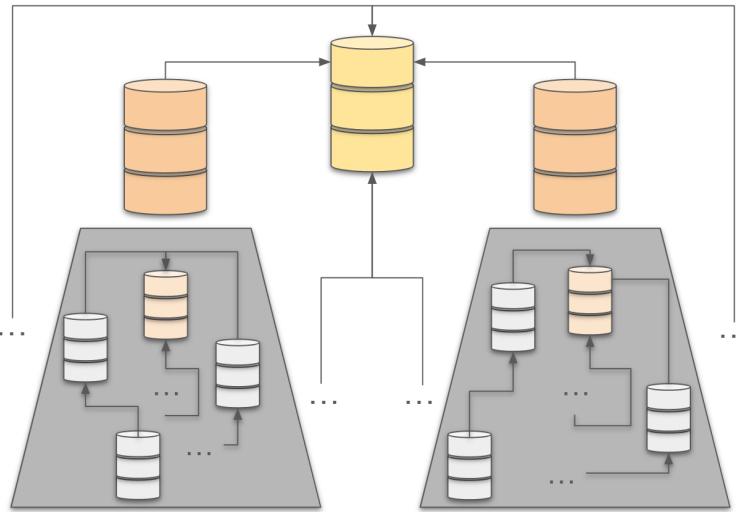


FIGURE 4.5: Different HDBs integrated into a larger unified DB.

#### 4.1.4 Data Quality

Data labels, if available at all, may not always accurately represent the actual ground truth and can be unreliable or vague. For instance, a particular issue with an entity could have multiple underlying root causes or error patterns, which may not match or correlate with the class labels. Some features and observations, particularly if they are *not* collected from *objective* sensor data but obtained in some other unreliable and uncertain way (e.g. textual data and reports), may not be entirely accurate, representative, or true, as they could be (considerably) influenced by subjective perceptions, biases, outliers, etc. This is especially true when working with textual data that is gathered from very different sources and people: it presents well-known (non-trivial) challenges inherent to natural language processing, including synonymy, antonymy, ambiguity, contextual meaning, grammatical accuracy, and more. Since ML models "learn" from data, the quality of the data has a crucial effect on the effectiveness and performance of the model; and poor data quality poses extra challenges. This holds true for both the label quality and the raw feature quality.

#### 4.1.5 Data Processing

Our objective is to detect (error) patterns in a very large collection of observations; which we will refer to as "defects". These defects are of great interest as they help us categorize patterns and identify the causes of problems, allowing for more in-depth analysis. However, a defect may have multiple underlying causes, including both known and *unknown* problems, making the scenario more complicated than simple multi-label classification. Our goal is to determine these, possibly overlapping, labels and defects (by learning useful embedding representations of the data objects and applying pattern recognition techniques to them). Once we have identified (potential) defects, we want to find general rules that match these patterns; allowing us to classify and organize these (previously unknown) defects. This helps us to classify and organize previously unknown defects. Once we have extracted new defect types (generalizable patterns), any subsequent new observation that does not satisfy

these (new, old) rules requires special attention and must be handled separately; it could be an outlier or a *new* (unknown) defect. In fact, if many new observations do not fit, it may even indicate that the extracted patterns are not well representative, incomplete, or that the data has changed too much (rendering rules obsolete).

Identifying patterns is important in the industry as it helps improve product quality, identify potential causes and correlations between defects, and prevent similar defects from occurring in the future. Not only does this reduce maintenance, and improve product robustness, reliability, and safety; but it can also enhance a company's reputation for quality, and increase customer satisfaction (hence, extending beyond immediate [quality] improvements; also encompassing long-term benefits).

#### 4.1.6 Problem Statement

The objective is to create a model that can analyze a dataset with different types of observations at varying sample rates, quantities, and qualities. The model should be able to identify important variables that influence the outcome, predict the result based on input variables, detect outliers, and locate similar objects. Moreover, the model should provide insights into the patterns of the dataset. This multifaceted problem, thus, entails addressing the following key components:

1. **Heterogeneous Data Management:** Efficiently handling diverse data types, particularly focusing on large hierarchical databases.
2. **Predictive Modeling:** Developing accurate prediction capabilities to offer valuable insights for new, future observations.
3. **Anomaly Detection:** Detecting and addressing anomalies that may impact analysis and predictions, and identifying (new, rare) patterns that require special attention.
4. **Similarity Assessment:** Identifying and grouping entities based on similarity, shared characteristics, or behaviors; facilitating comparison and analysis.
5. **Pattern Recognition:** Establishing generalizable rules and identifying data relationships influencing outcomes or displaying patterns.
6. **Interpretable Patterns:** Identifying important features (i.e. variables) and explaining inherent patterns, behaviors, and relationships; enabling informed decision-making and action.

#### Take-Away

Integrating diverse data sources and formats, such as sensor data and customizable reports, into a complex hierarchical database network structure and systematically organizing relationships between various entities and their attributes, is a major challenge. Ensuring data quality, especially when dealing with subjective information like textual data, is also a major hurdle to overcome. The ultimate goal is to analyze complex, unconventional data to identify patterns or anomalies and use them to improve quality, detect irregularities, reduce expenses, and enable the application of Machine Learning.

#### Keywords

Pattern & Anomaly Detection • Predictive Modeling • Heterogeneous Data

## 4.2 Previous Methods Employed in Real-World Applications

Following Chapter 4.1, this thesis will focus on automated decision-making and error pattern recognition by learning meaningful "embedding" representations (see upcoming Chapters 4.3 & 4.5). Prior to the start of this thesis, error patterns were identified through laborious, tedious, and time-consuming *manual* analysis of the data; trying to come up with general rules that would (so was the hope) systematically classify defects. In reality, however, such rules were mostly guided and motivated by heuristics and simple observations. The rules essentially comprised a collection of conjunctions and disjunctions of conditions. Generally, given a concatenation operator  $\otimes \in [\wedge, \vee]$  and a set of Boolean properties  $\psi \in \Psi$ , such rules can be equivalently reduced and rewritten in a Disjunctive Normal Form (DNF); every Boolean formula can be transformed into DNF (Pfahringer, 2017):

$$\text{Rule} := \bigvee_{\psi_i \in \Psi} \bigwedge [\neg] \psi_i \quad (4.4)$$

where  $[\neg]$  denotes an optional negation parameter. For example, one possible rule could be:

$$\text{Rule} \leftarrow [\psi_0 \vee (\psi_1 \wedge \psi_2) \vee (\neg\psi_0 \wedge \neg\psi_3)]$$

One benefit of having the rules represented in DNF is the ability to quickly verify in linear time complexity whether a given configuration satisfies any clause or not. This is because every clause in DNF operates independently via OR operations. To verify whether a configuration fulfills the rule, we can simply check if an individual clause applies. If the configuration meets any of these clauses, the entire rule (presented as a collection of clauses joined by disjunctions) is considered satisfied. Additionally, if we transform all rules to DNF, we obtain a unified rule-system template. Yet, the transformation of logical expressions into DNF may result in an exponential expansion of the expression's size and might not consistently generate the most efficient representation, considering that converting very complex logical expressions into DNF can be computationally intensive.

However, as mentioned earlier (Chapter 2.2.2 & 2.2.6), RBSs (i.e. rules in DNF) have many drawbacks and limitations; importantly: they do *not* determine any (e.g. semantic, meaningful) "embedding representation" of the data.

### 4.2.1 Concerns

Much like the limitations of RBSs we previously demonstrated for NLP tasks (see Chapter 2.2.6), creating these (DNF) rules presents equivalent problems. This holds regardless of whether the data being handled is textual or non-textual, such as tabular data. Devising such rules demands an extensive amount of time, involving the manual examination of raw data to discern patterns and subsequently formulating correct rules. Yet, these rules often fail to generalize effectively, potentially leading to the misclassification of defects and thereby further weakening the RBS. Even if possessing expertise in the field, independently formulated rules are easily subjective, influenced by personal and different experiences, considerations, observations, and heuristics. This subjectivity, lacking any rigorous mathematical validation, justification, or optimization, easily results in the creation of redundant or inaccurate rules, leading to numerous inconsistencies and problems.

To summarize, key shortcomings include: (1) time-consuming rule creation, (2) challenges in recognizing patterns from raw data, (3) ineffective rule generalization, (4) possible misclassification of defects, (5) subjectively formulated rules, (6) absence

of mathematical reasoning, (7) redundancy in rule creation, (8) inaccuracies and inconsistencies in RBSs, (9) lack of optimization, (10) scalability concerns, (11) maintenance difficulties, (12) conflicting rules and interdependencies, (13) limited adaptability in dynamic environments, (14) cost and resource intensiveness, and (15) risks of overfitting and underfitting; among others.

In the next section, we will discuss the advantages of using automated AI systems (particularly ML models), as a promising alternative and better solution.

#### Take-Away

Prior to this thesis, error patterns were manually identified, leading to rule-based systems based on heuristics and simple observations. These lacked generalizability and mathematical support. Consequently, the limitations of manual rules in defect classification prompted the exploration of Artificial Intelligence, especially Machine Learning, as a more promising alternative.

#### Keywords

Error Patterns • Rule-based Systems • Heuristics • Defect Classification

## 4.3 Machine Learning for Automated Error-Pattern Detection

The growing demand for automation is clear: aiming for scalable and efficient systems. However, while automation alone does *not* assure optimal solutions, leveraging ML principles enables us to automatically approximate an optimal solution in a mathematically sound way. ML approaches involve merging automation and optimization to create a model capable of effective (automatized, optimal) learning and prediction; eliminating the need for human intervention. Although the learning phase might be time-intensive, ranging from minutes to potentially weeks or even longer (depending on the model's complexity and dataset size), the prediction phase is very fast, typically occurring in real-time.

The primary objective can be defined as follows: developing an ML model capable of automatically understanding, learning, extracting, and extrapolating patterns from large databases. When presented with new, previously unseen data, this model should effectively apply the learned (generalizable) patterns and knowledge to generate new, accurate predictions.

ML involves several sequential stages: first, data *preprocessing*, e.g. to clean, normalize, and extract features from raw data, etc.; preparing it so for model training. Next, the ML model is *trained* to learn patterns and relationships given that preprocessed data. Hyperparameter optimization takes place during this training phase; thereby tweaking (i.e. adjusting) internal non-learnable parameters to improve the model's performance. Once fully (optimally) trained, the model moves to the prediction (or *inference*) phase; applying its learned generalized insights to make predictions to new, previously unseen data. Inference is commonly near instantaneous (while training may often require a substantial amount of time and resource power; depending on the model, data, desired effectiveness, etc.).

Figure 4.6 below illustrates, in a very simplified manner, the high-level idea of ML in a general, basic supervised setting (for automatic error-pattern detection); focusing on learning from observations and predicting outcomes for new instances. The three phases (pre-processing, training, and inference) were omitted for clarity.

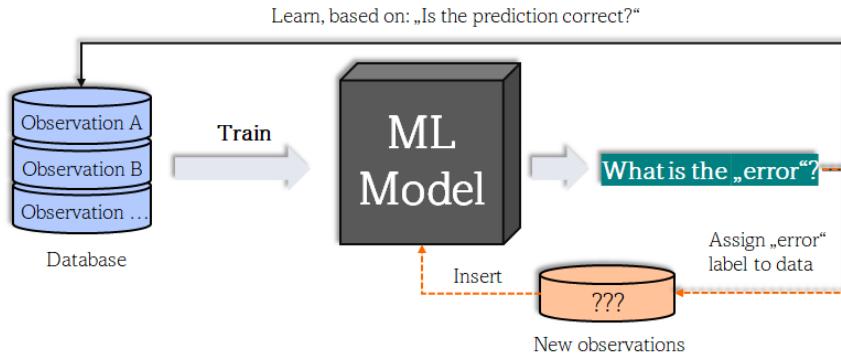


FIGURE 4.6: A simplified illustration demonstrating the idea of supervised ML for error-pattern detection; showcasing the process of learning from observations to predict outcomes for new instances.

### 4.3.1 Challenges in Machine Learning

Before we dive deeper into ML, it is important to understand the challenges of developing and deploying ML models (AI systems). While we will not further focus on these challenges in this work, it is important to keep them in mind to understand the drawbacks and risks of deploying and developing ML models. Moreover, these challenges are general and not exclusive to the field of identifying error patterns. Therefore, in this section, we will briefly summarize three key (yet often neglected) challenges in ML. This is by far not an extensive list and we encourage the reader to review the corresponding literature (L'heureux et al., 2017; Paleyes, Urma, and Lawrence, 2022; Zhou et al., 2017; Sze et al., 2017):

#### Expensive Development

Setting up serious ML models can be expensive and is usually more beneficial for long-term investments. To start, an ML team usually is composed of several distinct roles (e.g. Data Engineer, Data Scientist, Statistician & Analyst, Applied Machine Learning Engineer, etc.). Each role has its own specific application area, which makes ML a very interdisciplinary field. Many tasks arise implicitly when using ML, such as data gathering, data processing, model architecture search, model optimization, pipeline and workflow design, efficient, modular, and scalable implementation, setting up the right ML infrastructure, and many more. A low-level understanding of ML models requires a solid background in mathematics, statistics, and computer science; and technology changes and obsoletes rapidly. A high-level understanding is easier to obtain; yet lacking a detailed understanding restricts the ability to effectively adapt or apply ML techniques (and their many variants) to different practical use cases (domains). Overall, the development of ML technology and conducting respective research is expensive.

#### Data & Need for Custom Tailoring

Each use case is different, encompassing diverse types, quantities, and qualities of data, often carrying inherent uniqueness (e.g. challenges, objectives, constraints, complexities). While the overarching goal of ML is to enable autonomous (optimal)

learning without constant intervention from domain experts, ideally operating independently on the specific task at hand (i.e. not being *explicitly* programmed), each project presents its own distinct challenges.

In other words, variances in data, ranging from types and structures to the quantity and quality available, significantly influence the ML process and final model effectiveness. Hence, the pivotal role of data cannot be overstated. It plays a crucial role in ML systems and acquiring good data (or any substantial and relevant data) is often the first big challenge to overcome. A general rule of thumb is as follows: If the input is bad, so will the output. Or more informally but commonly stated among practitioners: "Garbage in, garbage out" or even "What you put in is what you get out". However, this is also a dangerously tempting justification for many individuals seeking to downplay model effectiveness: by simply blaming the data.

Essentially, the quality of the input directly influences the quality of the output. ML models are complex mathematical constructs; they do not possess magical "AI" capabilities. If the input lacks informative content or is full of obscured, noisy, or distorted patterns, the task of deriving generalized, applicable patterns becomes very challenging; if not outright impossible. Therefore, the type and quality of the input data greatly determine the success and efficacy of machine learning models. In many cases, a universal solution simply does not apply. Each use case tends to come with its unique set of constraints, limitations, or varying objectives. The ability to effectively tailor an ML model and fine-tune the data and learning procedures according to the specific demands of that use case is key; especially considering particular nuances inherent in each use case. Viewing an AI system as a singular black-box capable of handling any input, generating any desired output, and seamlessly adapting to every conceivable use case is fundamentally just wrong. Despite being obvious and apparent, this understanding is typically absent among non-AI practitioners.

### Reasoning, Uncertainty, and Reliability

**Notice:** Our work and findings on  $\alpha$ Max-B-CUBED (presented in this thesis) have been publicly released on OpenReview-Venue (Guimerà Cuevas and Schmid, n.d.).

ML is often viewed as an opaque black-box model. An input comes in, an output comes out. For those unfamiliar with ML and AI, questions like "*How does it work?*" or "*What drives its predictions?*" naturally arise. These black-box models are, in fact, highly complex and typically quite convoluted (even to individuals with strong AI backgrounds); posing serious challenges in AI reasoning and managing uncertainty. These challenges, thus, persist as active areas of research (Gawlikowski et al., 2021; Angelov and Soares, 2020).

Guaranteeing assurances regarding a model's performance is, therefore, non-trivial and difficult. Developing reliable, accurate, and comprehensive formal verification techniques remains an ongoing challenge (Huang et al., 2017; Leofante et al., 2018). The opacity of these models often also raises ethical concerns, e.g. about biases, especially in critical decision-making applications like healthcare and criminal justice (Li, Ruijs, and Lu, 2023; Smith, 2021). Model reliability is a very complex task that involves numerous further concerns; e.g. the notorious vulnerability of DL models to adversarial inputs or attacks<sup>1</sup>; which can cause a model to make incorrect (high-confident) predictions (Biggio and Roli, 2018). Overall, the interpretability of ML models and explainable AI are topics of great interest in this regard, i.e. to improve reliability and counteract uncertainty (Xu et al., 2019).

<sup>1</sup>Adversarial attacks are *not* relevant to our thesis but serve to highlight the complexity of ensuring model quality and providing reliable assurances. For more details, see Chakraborty et al., 2018.

**Take-Away**

Machine Learning (ML) automates pattern identification via optimized models, learning (generalizable) rules from large datasets; enabling effective (real-time) predictions on new data. Implementing ML presents challenges including cost, customization needs, and complexities in reasoning and reliability.

**Keywords**

ML • Error-Pattern Detection • Automation and Optimization • Challenges

## 4.4 Neural Networks Based on Complex Numbers with Weights Constrained along the Unit Circle

**Notice:** Our work and findings on complex-valued NNs (presented in this thesis) have been published in Guimerà Cuevas, Phan, and Schmid, 2023. It was inspired and builds upon our previous research (Guimerà Cuevas and Phan, 2021), but differs significantly as it has been substantially improved and modified (both the complex architecture and its mathematical framework); thus, constitutes novel contributions. In particular, this novel contribution has been accepted for publication, after peer review. The Version of Record is available online at: [https://link.springer.com/chapter/10.1007/978-3-031-33374-3\\_28](https://link.springer.com/chapter/10.1007/978-3-031-33374-3_28). Use of this Version is subject to the publisher's Manuscript terms of use. Therefore, please refer to Guimerà Cuevas, Phan, and Schmid, 2023 for specific details.

Deep Learning, the prevalent approach in Machine Learning, typically employs Deep Neural Networks (DNNs) that use real-valued (RV) weights to determine activation function outputs, thereby shaping decision boundaries between classes (Goodfellow et al., 2016; LeCun, Bengio, and Hinton, 2015). Non-linear activation functions are crucial to these models, as they prevent DNNs from collapsing into single-layer perceptrons, which are limited by linear decision boundaries, as exemplified by the notorious XOR problem (Rosenblatt, 1957).

This thesis details a novel technique for computing decision boundaries by integrating complex numbers into a bi-nonlinear neural network. Unlike traditional Complex-Valued Neural Networks (CVNNs), this approach uses RV weights as angles representing complex unit numbers, which are then scaled by the magnitude of the inputs and summed. This method offers several advantages over previous CVNN approaches: (1) Weights do not cancel inputs when set to zero, (2) Complex back-propagation is not necessary, (3) Neural inputs remain in the real domain, and (4) The weight matrix is RV. Additionally, this model avoids the need for explicit input mapping to complex phases and eliminates the risk of exploding gradients, making it easier to integrate into existing architectures without modifications (Aizenberg, 2011a; Amin and Murase, 2009; Philipp, Song, and Carbonell, 2017).

**Take-Away**

We propose a new technique using complex numbers in Neural Networks that computes decision boundaries without explicit input mapping. It preserves information, prevents input suppression, and seamlessly integrates into existing models without altering their architecture.

**Keywords**

Complex Numbers • Neural Networks • Non-linear Decision Boundaries

## 4.5 Deep Learning for Hierarchical Databases with Recursive One-to-Many Relations

In Section 4.1, we have presented examples of (real-world) use cases where databases (DB) are structured hierarchically and highlighted how individual entities across different DBs may exhibit different characteristics and features (both in type and quantity). This poses many challenges, making the application of standard DL non-trivial and not directly possible in such cases (i.e. data structures).

The application of DL to hierarchical databases (HDBs) is of great interest as it allows for the analysis of complex and large hierarchically structured datasets (these are commonly encountered; especially in the industry). HDBs are widespread and can be found in many different scenarios where data can be arranged in tree-like structures, with nodes representing entities or objects, and links representing specific relationships (such as dependencies or connections between them). While DL is typically used to analyze individual entities *within* a database (DB) table, there is a growing need to explore relationships between entities across *different* tables and DB network structures. Extracting complex patterns and insights from a DB and its hierarchical interconnections can greatly enhance understanding, particularly in terms of its topological HDB structure (determined by the [relational] links between HDB entities). This, combined with the individual features of each HDB entity, can be highly useful for various tasks, especially if a meaningful semantic single vector representation is required; as is often the case for downstream applications.

Semantic representations can be obtained through Embedding Learning; which is an important concept in ML, used to represent data as continuous vectors. "Embeddings" are *semantic* mathematical representations that transform data into a vector space (typically, of fixed dimension). Often, embeddings are relatively low-dimensional representations of objects in high-dimensional spaces that *distill* inherent information (not the same as compression; see Chapter 2.3); enabling so efficient analysis and further processing, e.g. for subsequent downstream tasks. They typically capture (different) semantic meanings of inputs (often in relation to other inputs) and can be learned automatically by a model. The main idea is that similar items are represented by similar vectors; which helps to capture meaningful relationships between items. Embeddings can often be (and are) reused across different models and are especially renowned in the context of natural language processing (NLP) (Almeida and Xexéo, 2019), typically in conjunction with word embeddings; including respective evaluation methods (Bakarov, 2018). Similarly, graph embeddings (Goyal and Ferrara, 2018) and network representation learning (Zhang et al., 2018) aim at "embedding" graphs into low-dimensional vector spaces while preserving network information (e.g. vertices, topological structure, etc.). Embeddings can be used for retrieving similar objects (Hjaltason and Samet, 2003), zero-shot learning (Zhang and Saligrama, 2015), anomaly detection (Hu et al., 2016), graph problems (Cai, Zheng, and Chang, 2018; Goyal and Ferrara, 2018), clustering (Xu and Wunsch, 2005), model predictions (Adya and Collopy, 1998), and for many other DL applications. Finding "good" (latent) embedding representations for instances is, thus, of great importance for many different downstream tasks, since effective embedding

representations can improve model performance by capturing the inherent structure of the data more accurately. In other words, embeddings can simplify data dimensionality and complexity, which facilitates processing and evaluation; and, therefore, play a key role in many DL systems and applications.

"Pre-trained" embeddings are pre-existing numerical representations that encode semantic meaning or context (typically pre-trained on large datasets and models). They facilitate efficient learning for various tasks by either fine-tuning the respective pre-trained ML model or directly adapting the embeddings. This saves computational time by leveraging existing knowledge instead of training from scratch, simply utilizing pre-trained embeddings and continuing learning from there. Meaningful pre-trained embeddings often contribute to improved generalization by distinguishing between similar and dissimilar objects, preserving relationships or individual characteristics, and capturing semantic nuances within the data.

Although DL on DBs is becoming increasingly more relevant for many applications, it still poses many non-trivial challenges. There are only relatively few approaches to using DL methods for solving (common, basic) DB problems (Wang et al., 2016b); especially in comparison to other fields in AI. DL on relational DBs is considerably more difficult and differs greatly from traditional DL methods on tabular data, images, natural language, etc. in the way the input data is structured and defined. Large-scaled DB systems are also computationally hard to analyze and process efficiently, such that special distributed infrastructure and techniques are often needed for good performance and efficient analysis, exploration, and processing (e.g. parallel training or multitenant resource scheduling) (Mayer and Jacobsen, 2020). HDBs are special types of DBs where relations implicitly form a hierarchy holding a root instance and loop-free recursive 1:n connections to further sub-tables. Each table may store different features and have further sub-tables. Recursive HDBs typically have more than one hierarchy level. They are commonly found in compositional settings, e.g. where a piece may be made of many sub-pieces, and each sub-piece can be recursively composed of an arbitrary number of further sub-pieces. Finding representative embeddings for HDBs is key to many downstream tasks.

In traditional DL settings, data takes a straightforward form such as numeric values, textual data, pixel images, video, audio, etc. For these types of inputs, there are already successfully established architectures and basic models that can work directly on these types of inputs. In these cases, the focus is often mainly on the model architecture for effectively performing downstream tasks and optimally generating embedding representations. However, for DBs (especially for HDBs), it is not possible to simply feed the DB into a model and somehow obtain embedding representations. The forward pass function of a traditional neural network processes data in the form of a fixed-size linear vector. However, HDBs have a complex graph structure, with varying features, topology, and size, where entities are connected through recursive one-to-many relationships. Representing such data in a linear format is not easy or even optimal and may require a more complex input structure and data processing strategy. Therefore, traditional neural architectures are *not* suitable for processing graphs, as they are designed for vector data (not graphs). Thus, it is necessary to map the DB to a knowledge representation that can be processed by the model in the first place. This step alone is challenging and fundamental. Poor input representations can destroy underlying data patterns, worsen data encoding quality, and lose information and potentially important data correlations (therefore, strong representations are essential). In other words, effectively converting a heterogeneous HDB into a structured knowledge representation or appropriate data format is essential for maintaining data integrity and enabling efficient processing.

Equally crucial is the processing function (i.e., the mathematical definition) of the corresponding DL model.

This thesis proposes an efficient and scalable method for applying DL to HDBs. It enables DL-based optimization (on HDBs) and can be used to learn representative embedding vectors (which can then be utilized for various downstream tasks; as is typical in DL). The different DB hierarchy levels can be automatically and independently weighted to improve overall model training and performance. Our recursive architecture can handle any number of hierarchy levels, and it integrates the underlying HDB structure directly into the model. The main novelty of our method lies in its definition of the model's forward-pass, architecture, and training strategy for processing HDB instances. It allows for an effective, direct processing of HDBs, without compromising data quality during input transformation and forward processing. Moreover, the proposed approach offers flexibility in handling different feature types (i.e., multimodal data).

#### Take-Away

Deep Learning (DL) can be applied to support Hierarchical Databases (HDBs), which organize data in tree-like structures. Embedding Learning, an important Machine Learning technique, converts data into continuous vectors for tasks such as retrieval, anomaly detection, model predictions, etc. However, applying DL to HDBs presents many challenges in terms of data processing, effectiveness, and computational costs; yet, DL on HDBs is very relevant for many real-world use cases.

#### Keywords

Hierarchical Databases • DL • Embeddings • Representation Learning

## 4.6 Determining Representative Textual Labels for Clustering Accurate Sensor Data with Inexact Annotations

Imagine a dataset containing information from sensors and textual sources. The sensor data typically offers accuracy and objectivity, whereas textual data may vary in quality: being inaccurate, imprecise, or irrelevant. When seeking patterns within sensor data, clustering directly using this sensor data alone proves effective (since sensor data is precise and reliable). However, additionally including textual data within the clustering procedures can be problematic due to potential noise and inaccuracies, especially when the textual data quality is low, noisy, etc. This can significantly affect the accuracy of clustering, making it harder to identify meaningful patterns and even destroy, obscure, or hide patterns of interest that would otherwise be found if we were clustering on the sensor data alone.

Yet, there may still be reasons and motivations to use the textual data for other specific purposes, such as identifying the most representative textual comments for the identified sensor clusters. However, this again poses several further difficulties since the textual data does not necessarily match the sensor data; it can/should, but does not necessarily have to. The most representative sensor instance for a cluster, such as the data instance closest to the cluster centroid, may e.g. be a low-quality textual annotation. As a result, a key and central issue is assessing whether the textual annotations are suitable and indicative of the total cluster regarding the sensor

data. We will refer to this as: clustering accurate sensor data with inexact annotations. "Inexact" here refers to the uncertainty and variability in the quality levels of individual annotations; potentially (but not necessarily) causing deviations from the the underlying objective sensor data.

Consider the following sensor data: {"LIGHT": 982, "SOUND": 55.7, "UV": 3, "CO2": 560, "MOTION": TRUE}. Several annotations could be given, among which may be of poor quality or unrelated. Some possible textual annotations could be:

1. *"The indoor environment is comfortable, but the slightly elevated CO2 levels could be an area of concern to improve ventilation."* [Good Quality]
2. *"Elevated CO2 levels; requires attention to ventilation."* [Moderate Quality]
3. *"Nothing special; I think, couldn't find anything, but unsure."* [Low Quality]
4. *"I don't like this job, I will quit."* [Off-topic/Low Quality]
5. *"The light level is at 982, suggesting the area is well-lit. The sound level is 55.7, which is pretty low. The ultraviolet index is at 3, indicating that the level of radiation is minimal. The gadget is also detecting the carbon dioxide (CO2) level, which is at 560 ppm, suggesting that the air quality may be improved [...]."* [Good Quality]

Hence, textual data may be unreliable, but sensor data is considered objective and can be clustered with reliable confidence. Therefore, clustering (solely) on textual data may well be problematic because it may identify patterns that do not represent or align with the sensor data; that is, it will find patterns within the textual data but not necessarily corresponding to the sensor data of interest. Combining low-quality textual data with high-quality sensor data for clustering can, thus, result in inaccurate results, as we may be adding low-quality data to the high-quality sensor data during clustering. Consequently, it makes sense to evaluate the quality of the associated textual annotations and identify the most fitting or representative ones that align with the sensor clusters to filter out and identify high-quality annotations against low-quality annotations. The fundamental questions are: "Which annotation best captures the essence of the cluster?" and "What distinguishes the vast majority of cluster instances?". This can be addressed by first adopting a probabilistic viewpoint assessing it differently and asking for: *"What annotation is most likely to act as the best (possible) replacement?"*; and similarly *"What is most likely to distinguish the vast majority of cluster instances?"*. Finding the best representative textual annotation, however, might be difficult in such inexact setting, especially if the clusters have many possible (different) interpretations. As a consequence, we will define "the best cluster representation" as the annotation with the **highest probability** of being the most representative combination of sensor- and textual data for the cluster instances (in comparison to the other available annotations); the respective scores will be used to establish a ranking of representation quality. We will do this by determining and calculating scores that balance the representative quality of sensor data against the quality of textual data; and produce a quality ranking with a focus on extracting top-quality representations.

**Take-Away**

Combining sensor data with textual data in clustering presents a challenge due to the objective nature of sensor data compared to the subjective and noise-prone characteristics of textual data. An issue arises when aiming to extract the most relevant textual annotations for clusters identified by sensor data, as poor-quality annotations can distort cluster patterns.

**Keywords**

Clustering • Textual data • Sensor data • Annotations • Quality Ranking

## 4.7 Explainable and Interpretable AI

**Notice:** Our work and findings on  $\alpha$ Max-B-CUBED (presented in this thesis) have been publicly released on OpenReview-Venue (Guimerà Cuevas and Schmid, n.d.).

### 4.7.1 Supervised Cluster Evaluation Metric

The evaluation of clustering methods and their results is a complex task due to the lack of clear-cut criteria for determining the quality of clusters (Rand, 1971). While clustering may seem simple in theory, it is difficult to create a general framework that works for all cases (Rai and Singh, 2010; Xu and Tian, 2015; Berkhin, 2006; Xu and Wunsch, 2005). In fact, in "An Impossibility Theorem for Clustering" (Kleinberg, 2002)", an example of three simple properties was presented for which no clustering function could satisfy all three at the same time, exposing inevitable trade-offs. Follow-up work (Ben-David and Ackerman, 2008) proposed considering clustering quality measurements as the object to be axiomatized rather than clustering functions and proposed a revised set of criteria (axioms) for such measures. The authors show that the *clustering-quality* framework is richer and more flexible than *clustering functions* because it allows the postulation of axioms that capture the features expressed by Kleinberg's axioms without producing a contradiction. Evaluation of clustering methods is thus important, due to the difficulty of developing a unified clustering framework that is independent of any underlying algorithm, objective, or model. Approaches to formalizing such qualitative objective criteria are mainly distinguished between two categories: *Intrinsic* and *Extrinsic* metrics. Intrinsic methods rely on inherent properties of the clustering results, while Extrinsic methods use external ground truths to infer the quality and effectiveness of clustering results. Such ground truth can e.g. be the labels of the data instances. Overall, evaluating the effectiveness of clustering methods and their outcomes remains a complex task.

In the earlier section (Chapter 2.10.2), the discussion centered on *extrinsic* clustering evaluation metrics and  $B^3$  (Amigó et al., 2009); where formal parameters for evaluation metrics were highlighted, emphasizing intuitive comprehension, limitations clarification, formal provability, and differentiation among metric families based on mathematical principles. To recap, four constraints on quality measurements were introduced: homogeneity, completeness, rag bag, and size versus quantity:

- *Homogeneity*: Clusters should not contain items from different categories.
  - E.g., a cluster of flowers should not mix with non-flowers.

- *Completeness*: Items from the same category should be clustered together.
  - E.g., different types of flowers should be together in a single cluster, not spread across different (non-flower) clusters.
- *Rag Bag*: Disorder should be less detrimental in a disordered cluster than in a clean cluster.
  - E.g., plants that do not fit well with other clusters can have a separate "miscellaneous" cluster; rather than forcing them into other clusters.
- *Cluster homogeneity*: A small error in a large cluster should be preferred over a high number of small errors in small clusters.
  - E.g., misclassifying some flowers into a *large* tree cluster is less impactful than having some misclassifications of trees spread across numerous *smaller* non-tree clusters.

Again, it was demonstrated that among commonly employed metrics, only the extrinsic  $B^3$  metric meets all four constraints, while others do not. For a graphical illustration of these constraints, refer to Amigó et al., 2009.

Clustering has two main components: the clustering *method* and the data *representation*. Different clustering algorithms identify different patterns and subgroups because they have different concepts of neighborhoods, assumptions about data distribution, strengths, and weaknesses, or use different distance or similarity metrics and interpretations. When the data representation is fixed, different clustering algorithms will produce different results, each with its own degree of success. For example, K-means (Ahmed, Seraj, and Islam, 2020) is a well-known and commonly used clustering algorithm noted for its simplicity and efficiency, particularly when dealing with spherical, well-separated clusters. Hierarchical clustering (Murtagh and Contreras, 2012) builds a hierarchy of clusters by merging smaller clusters into bigger ones and may find clusters of any shape, generating either hard or soft clusters. DBSCAN (Schubert et al., 2017) is a density-based clustering approach that discovers clusters of various forms and is resistant to outliers. Gaussian Mixture Models (Reynolds et al., 2009) are probabilistic models that assume the data is created by various Gaussian distributions, resulting in soft clusters that work well for data with complex cluster forms. Spectral Clustering (Von Luxburg, 2007) finds clusters by using the eigenvectors of a similarity matrix and is suited for data with a non-linear structure that is difficult to separate using linear approaches. Selecting the most appropriate clustering algorithm can then be done through appropriate evaluation, and therefore the correctness of the evaluation method is key here. On the other hand, a different perspective is to have a fixed clustering algorithm, but have the data representation be trainable, e.g. using different and trainable deep embedding of NNs for "deep clustering" (Zhou et al., 2022a; Caron et al., 2018; Bo et al., 2020), where a model learns how to best and optimally project and generate an effective data representation; i.e. given a fixed clustering algorithm.

For inexact labels in a weakly supervised context, the clustering evaluation of the model should account for this uncertainty. Another concern of employing a sub-optimal cluster metric is over-optimizing that (sub-optimal) metric e.g. without considering the true structure of the data; in the case of coarse labels, this means without considering sub-labels. A poor metric choice can also bias the evaluation towards a given number of clusters rather than representing the true structure. This is especially important for imbalanced datasets.

$B^3$  can be seen as a "precision and recall metric" for clusters. Following Amigó et al., 2009, two items that share a category are correctly related if and only if they occur in the same cluster. An item's  $B^3$  precision is the fraction of objects in its cluster that have the item's category. The average precision of all items in the distribution is used to calculate the overall  $B^3$  precision score. The  $B^3$  recall is analogous. Utilizing the  $B^3$  algorithm enables a numerical evaluation of the clustering assignments' quality. However, it should be noted that the  $B^3$  algorithm does not factor in imbalanced datasets; i.e. datasets where the distribution of classes or categories within the data is skewed and unequal.

$B^3$ , however, makes a very strong assumption: It presupposes that the ground truth labels are exact and that there are no (relevant) sub-clusters inside groups of equally labeled instances. As an example, suppose a data collection has  $n \cdot m$  labels, but these labels are not visible or known in advance. Instead, the visible labels are  $m$  groups containing  $n$  items in pairs, but one is still interested in discovering (i.e. identifying) the unknown number of sub-clusters  $n$ . Say, two clustering algorithms are given,  $C_A$  (which finds the  $m$  super-sets) and  $C_B$  (which finds the  $n \cdot m$  subsets). Although method  $C_B$  is obviously the preferred one, the  $B^3$  algorithm would favor algorithm  $C_A$  since  $C_B$  has a lower recall per cluster ("completeness"). As a result, one can argue and consider that the attribute "completeness" is problematic in such circumstances, meaning that an extrinsic assessment using the coarse labels as the reference can be problematic here. That is, the constraint that "different clusters should contain items from different categories" (Amigó et al., 2009) can fail to select the right model here when using the conventional  $B^3$  metric. Now, suppose there were no more subgroups and the true labels were the aforementioned  $m$  labels. In such a case, one would obviously want a measure that favors model  $C_A$  over model  $C_B$ . A fundamental challenge is, therefore, to construct a metric that provides a fair assessment of the clustering quality in both balanced and imbalanced data sets while adjusting for ground-truth label uncertainty. This is especially challenging when the true structure and nature (ground truth) of the data are unknown.

Yet, it is still possible to take advantage of a simple overall observation. Because of not having any 100% reliable way of comparing to ground truth, the process of breaking a set of clusters into multiple newer ones is fraught with "uncertainty." Assuming  $m$  (possibly sub-) clusters and a deterministic aggregation function that unifies clusters into super-sets, by then grouping clusters together into (less) super-clusters, one can now move from "uncertainty" closer to "certainty" since it allows one to conclude: If the newly grouped clusters were of high quality, then super-clusters are also "more likely" to be of high quality. By contraposition, if the super-clusters are "less likely" to have good quality, i.e. are of lower quality, then so are the sub-clusters (Figure 4.7).

Based on this motivation, we can propose a new method to evaluate the clustering quality to address the term "completeness" in the conventional  $B^3$  metric.

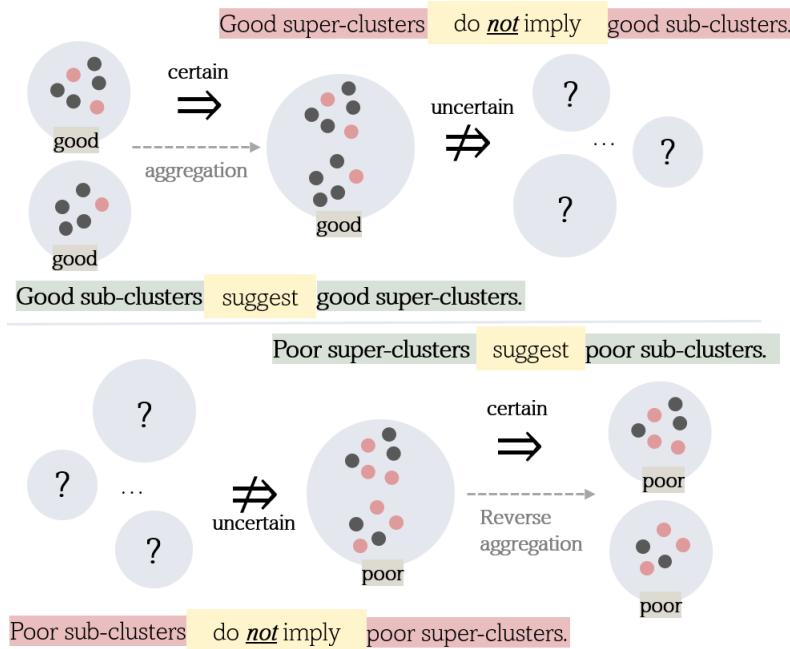


FIGURE 4.7: Quality implications based on deterministic and non-deterministic cluster aggregations and separations.

### Take-Away

Evaluating clustering methods can be difficult due to the lack of universally accepted criteria for measuring cluster quality across various scenarios. Each clustering algorithm has its own strengths and weaknesses, leading to different evaluation metrics based on the characteristics of the data and analysis objectives. This requires exploring both intrinsic and extrinsic metrics and addressing constraints to improve evaluation methods. To meet several quality constraints and reveal how algorithm characteristics affect success rates, the B-CUBED ( $B^3$ ) metric has been a focus. However, there is criticism about  $B^3$  for its limitations in assessing imbalanced datasets and handling uncertainty in clustering subgroups. However,  $B^3$  is a supervised metric, but for unsupervised problems, different metrics are employed.

### Keywords

Clustering • Evaluation Metrics • B-CUBED • Algorithm Characteristics

## 4.8 Automated Textual Description Generation of Clusters

Clustering, as already motivated, is a fundamental concept in Data Mining and ML; and particularly in DL. Its primary goal is to group objects that share similarities (Berkhin, 2006; Caron et al., 2018). A multitude of different clustering algorithms are available, including methods based on cluster centroids, data density, data distribution, and data hierarchies. The key challenge centers around accurately defining the notion of "similarity" and also in identifying suitable representation vectors

for the data. Advancements in NN-based approaches have demonstrated remarkable effectiveness in both data representation learning and dimensionality reduction, particularly in capturing visual similarities within images (Bengio, Courville, and Vincent, 2013; Hinton and Salakhutdinov, 2006). Similar data representations, or representations/embeddings close in the manifold space, typically indicate some sort of (semantic) similarity, commonality, etc.

In the field of image captioning, DNNs are often used to generate a textual description of an image utilizing VisionEncoderDecoder transformer models (ViED) (Li et al., 2021b). These models combine pre-trained vision encoders, such as ViT (Dosovitskiy et al., 2020) and Swin (Liu et al., 2021), with pre-trained language model decoders like GPT2 (Radford et al., 2019), Llama2 (Touvron et al., 2023), or other LLMs to guide the generation of an auto-regressive language model. Training is typically (very) resource-intensive and requires powerful hardware and also substantial amounts of labeled image data. Consequently, pre-trained models are often used and often (partly) frozen and/or fine-tuned for specific tasks.

Recall, that manifold embeddings are used to represent high-dimensional data in a more compact, lower-dimensional space, i.e. a subspace contained within the feature space (Li et al., 2022). This enables the grouping of data points according to their proximity on the identified manifolds, rather than their positions in the input feature space. DNNs are particularly well-suited for generating semantic manifolds in general; and ViEDs in the context of image captioning.

Text decoding strategies are used to generate text from a language model by identifying the most likely word sequence among potential outputs, and selecting the next tokens based on the context of preceding words. Several common decoding strategies exist (Wolf et al., 2020b), like Greedy-search, which selects the word with the highest probability at each time step, or Multinomial sampling, a method that randomly selects the next token according to the model's probability distribution. Beam-search maintains multiple potential sequences at each step, while Beam-search multinomial sampling combines beam-search with multinomial sampling. Diverse beam search focuses on generating a diverse set of beam sequences by optimizing for diversity-augmented objectives. Sampling strategies can also be used to stochastically generate multiple sequences ( $k$ -sequence sampling).

Population-based sampling involves selecting a representative sample from a larger population, where sample candidates are chosen from a diverse pool of potential solutions to a problem. The function used to assess the quality of each sample is known as the score-function (Bäck and Schwefel, 1993): a specific type of objective function that quantifies how effectively a given candidate solution achieves a desired or optimal goal. This is valuable when a model is trained to optimize a differentiable objective loss, which acts as a proxy for or approximates another non-differentiable metric loss of interest. In such cases, multiple candidate solutions may be generated, and the one with the highest metric loss is selected.

In this work, we demonstrate that, already by employing simple statistical and geometric methods to combine latent states of instances within a cluster, we can obtain a generative model for generating concise cluster descriptions. In particular, when used with a population-based decoding sampling method ranked by a score function, the model can produce even more representative and concise cluster captions that closely match the textual content of all individual image captions within the cluster on average.

**Take-Away**

Clustering is the process of grouping similar objects based on their similarities. Deep Neural Networks are used in image captioning through VisionEncoderDecoder (ViED) transformer models. These networks are effective in learning data representations and reducing dimensions, particularly in capturing visual similarities in images. A generative model for generating concise cluster descriptions can be developed using simple statistical and geometric methods operating directly on the respective ViED manifold space.

**Keywords**

Clustering • Algorithms • DNNs • Image Captioning • Transformer Models

## 4.9 Non-linear Normalization of Heterogeneous Feature Distributions with Adaptive Tanh-Estimators

**Please note:** This contribution has been accepted for publication, after peer review in Guimerà Cuevas and Schmid, 2024b. The Version of Record is available online. And so, our following work and findings on Tanh-Estimators (detailed in this thesis) were presented by us in conference proceedings (Guimerà Cuevas and Schmid, 2024b).

Knowledge representation (KR) is the process of arranging and organizing information such that a computer can interpret and manipulate it. The goal of KR is to express information, concepts, and associations in a precise manner that a computer can understand. There are several ways to represent data, with feature vectors being one of the most typical for many applications and data types. A feature vector is a numerical representation of a data instance in which each feature relates to a different attribute or characteristic of the instance. Feature scaling (FS), also known as data normalization, is a data pre-processing technique that is used to adjust the range and scope of variables (i.e. features). Effective pre-processing of numeric data- and knowledge representations is crucial for good model performance, as the ranges of feature values can vary widely, yet comparable domains are usually essential for optimal training. When it comes to training time, convergence, and performance, adequate FS might be the difference between generating a strong or weak model, or even the difference between complete failure and success. Furthermore, especially for neural networks, gradient descent converges much quicker with FS than without it. Different types of data (e.g. tabular data, images, sound waves, time series, text, etc.) may support different normalization techniques. Non-linear trend removal, for example, does not apply to textual data. Methods on tabular data, however, are commonly widely applicable since they typically operate on numerical values, which are prevalent in most data domains. Re-scaling through min-max normalization, mean/median normalization, standardization (alias Z-score normalization), and scaling to unit length are some of the most popular methods for tabular features. Other non-linear transformations include translations to uniform distributions (e.g. with quantile transformations) or Gaussian distributions (e.g. with power transformations), and polynomial feature generation. There are numerous more methods, such as data discretization, -clipping, or log-scaling. FS is a sub-field of data pre-processing and is considered its own building block, separated from related concepts such as data cleaning, - transformation, -integration, noise detection,

or missing value-imputation (García et al., 2016). Data pre-processing and feature normalization are well explored, and there are several extensive reviews available in the literature (García, Luengo, and Herrera, 2015; Subasi, 2020; Jayalakshmi and Santhakumaran, 2011; Li et al., 2021a).

The domain range and distribution of feature values are, hence, crucial in machine learning. Outliers are data points that deviate considerably from the rest of the observations in a particular data collection (Miller, 1993; Chiang, Pell, and Seasholtz, 2003; Rousseeuw and Hubert, 2011), e.g. extreme statistical points at the tails of feature distributions or points that come from a completely different distribution (out-of-distribution). Outliers can have a negative impact and mislead the training process, resulting in longer training times, poorer generalization, inferior performance, or the violation of statistical assumptions. There are many different types of outliers (Foorthuis, 2018). They are classified as univariate or multivariate, i.e. independent or dependent on other features, and further divided into three categories: point anomalies, conditional outliers, and collective outliers. Point anomalies (a.k.a. global outliers or simple anomalies) are simply individual data points that stand out and are e.g. exceptionally different from the rest. They exist independently and do not rely on any specific condition or context to be considered unusual. E.g. a single sunflower in a field of shorter flowers stands out due to its significantly greater height. Conditional outliers (a.k.a. local outliers or conditional anomalies) stand out within specific subsets or conditions of the data, but might otherwise not be outliers from a global perspective. E.g., within a group of the same plant species, one plant may display slower growth than the rest when placed in a specific environment. Collective outliers (a.k.a. contextual outliers) involve a group of data points that "together" collectively stand out from the rest of the dataset. In other words, individually, these points might not be considered outliers, but together, they form a subset that is (significantly) different. E.g., a sudden cluster of equally-colored flowers in a garden full of many vibrant-colored (different) flowers stands out; however, individually these flowers would not.

The *Tanh*-estimator (TE) normalization, first introduced by (Hampel et al., 2011), aims to mitigate and suppress univariate outliers for *point anomalies*. This normalization technique employs a tanh function in conjunction with a fixed spread factor on feature values that have independently been pre-scaled feature-wise (Equation 7.4); squishing and bounding extreme values inside a desired range. Prior to the actual TE normalization, the pre-scaling stage utilizes the *Hampel* function (Hampel, 1974; Shevlyakov, Morgenthaler, and Shurygin, 2008) for robust estimation that minimizes the influence of anomalies by lowering the importance of points towards the distribution's tails. A trade-off between robustness and optimality can be imposed by changing the function's hyper-parameters e.g. to produce stronger or weaker effects on the distribution tails. The tanh spread factor parameter, on the other hand, influences the standard deviation of the pre-scaled feature distributions, stretching or contracting them respectively. In Latha and Thangasamy, 2011, a modified tanh-normalization was proposed, wherein the raw features' mean and standard deviation were used instead of the Hampel function.

However, by employing a (single) normalization spread value that ultimately represents a constant, fixed manipulation parameter that is applied to all features, TE essentially introduces a new hyper-parameter that needs manual adjustment, experimentation, tweaking, etc. Thus, incorrect or unfortunate spread values may squish or stretch the value distribution excessively hard/soft, which may be harmful to training. As a result, the idea of having a fixed spread value for all features with

possibly even very different distributions is questionable. That is, given large feature input dimensions, manually tuning and estimating the spread value of features may be very time-consuming, tedious, impracticable, or at a certain point simply unfeasible, i.e. non-scalable.

In this thesis, we propose and introduce the heterogeneous Wasserstein tanh-estimator (WTE), an adaptive form of tanh-normalization that automatically estimates the ideal spread factors that minimize the *Wasserstein Distance* (WD) (Panaretos and Zemel, 2019) of different heterogeneous feature distributions versus a target distribution (e.g. Gaussian). WTE builds upon and improves upon follow-up work Latha and Thangasamy, 2011 on the tanh-normalization, which applies the tanh directly on the standardized features without applying the *Hampel* estimator for efficiency reasons. Overall, WTE can clearly enhance the convergence speed compared to relying on a single fixed default spread-value (e.g., 0.01), particularly in the initial phases of training. Our method's essential qualities are as follows:

- It minimizes the distribution disparity between normalized features and a target distribution to prevent poor or sub-optimal feature distributions.
- It requires no hyper-parameter tuning and finds the ideal spread value via simple scalar minimization methods (e.g. *Brent* (Brent, 1971)).
- The pre-processing can (optionally) be incorporated into the model's training loss and optimized accordingly.

For instance, it's often advantageous to aim for output distributions closely resembling a Gaussian distribution across all features originating from diverse value distributions. In this context, WTE efficiently identifies the optimal spread value for each feature, aligning them with the Gaussian distribution as closely as feasible. By minimizing discrepancies, this approach better ensures a more effective initial training phase, mitigating the need for weight readjustments as a result of weak or suboptimal FS. Therefore, the model is better positioned to quicker and easier learn from the data.

Aiming for Gaussian-style distributions can be advantageous in various situations and is intuitive, but it might not necessarily be suitable for all datasets or models. Hence, in this work, we employ the Gaussian distribution as an example distribution while framing this work within the concept of a "target distribution."

### Take-Away

Knowledge Representation organizes data for computer understanding. When dealing with numerical data represented as feature vectors, it's typically necessary to scale adequately them to improve model performance. Outliers, representing extreme data points, can harm the training process. The *Tanh*-estimator normalization technique handles outliers, but assigning uniform spread values to heterogeneous and different feature distributions might hinder the training progress. Instead, we can leverage the Wasserstein Distance to automatically determine the (different) optimal spread values. This enhances training convergence by reducing differences between normalized features and target distributions; all without the need for manual tweaking.

#### Keywords

Feature Scaling • Preprocessing • Wasserstein Distance • Outliers • Tanh

## 4.10 Transferring Knowledge Across Input Domains: Distilling Insights from Machine Learning Models Trained on Different Datasets

Real-world data is often intricate and complex. Analyzing it by hand can be tiresome and imprecise. This can lead to only a small fraction of easily understandable data being looked at, i.e. to only some of the data being analyzed at a given time or even at all. And so, this often leads to prioritizing general and abstract information while overlooking important but less obvious (i.e. more specific) data. For example, manual categorization may only be conducted on the more general and abstract intuitive data, while the more specific data is left unanalyzed. Furthermore, the integration of new sensors or hardware can potentially unveil previously inaccessible data, which had not been accounted for in the initial analysis; but might be too complex for intuitive human understanding. Old data may also become outdated or unobtainable (i.e. not available or accessible anymore) e.g. due to changes in environments, updated tools (like software modifications), changes in laws, restrictions, etc.

For example, a specific use case can be in predictive maintenance for industries such as manufacturing. There may be different data sources that can be exploited to train ML models. By combining insights from different data, a more accurate and reliable model may be built.

Hence, by distilling and building insights from different data- and knowledge sources, a better and more reliable model can be constructed; in particular by building upon previously extracted and learned knowledge. This aligns with the idea of transferring knowledge among input/output domains. It involves harnessing insights and information obtained from diverse datasets or sources, e.g. to enhance the effectiveness and resilience of other ML models.

#### Take-Away

Manually analyzing complex real-world data is slow and can overlook nuanced details, e.g. resulting in an incomplete analysis. Employing Machine Learning on different data sources may be either necessary and crucial in cases of limited access to old data, or of explicit interest e.g. to allow for more effective predictive models despite being more challenging; leveraging different dataset insights and prior knowledge can enhance model accuracy, reliability, and training efficiency by capitalizing and building upon already established knowledge and facts.

#### Keywords

Knowledge Transfer • Complex Data • Distilling Insights • Challenges

## 4.11 Model Calibration for Out-of-Distribution Detection

**Please note:** Our following work and findings on Calibration Misalignment as a Post-hoc Approach for Out-of-Distribution Detection in Deep Neural Networks (presented in this thesis) have been accepted for publication in conference proceedings (Guimerà Cuevas and Schmid, 2024a).

Calibrating the output of a statistical or ML model is a common step for fine-tuning its predictions to more accurately reflect the true probabilities of the events it predicts on average (Vaicenavicius et al., 2019; Bella et al., 2010). The goal is to achieve a "well-calibrated" model that reflects the observed distributions in the data, ensuring reliable and trustworthy predictions by mitigating both overconfidence and underconfidence. For example, in a binary classification model predicting an outcome based on input features, rounding output probabilities to 0 or 1 can result in inappropriate accounting for prediction uncertainty. Overly confident scores may be assigned to marginally probable instances and excessively cautious scores to instances with probabilities close to but below the decision threshold. Overall, the classification accuracy of a classifier remains unaffected and is not influenced by the calibration process as long as the ranking of the class probabilities is maintained.

It has been observed that modern NN models often exhibit poor calibration, with increasing classification accuracy corresponding to worsening probability miscalibration error (Guo et al., 2017). Thus, model calibration becomes a valuable method to ensure well-calibrated probabilities for predictive ML models and -systems. However, it is important to recognize that the calibration process can be laborious and may not always be necessary depending on the specific application.

Model calibration in predictive models can fail in different scenarios, but there are diverse methods available to assess and monitor the calibration quality (Kuhn, Johnson, et al., 2013). If a model is overfitting the training data, it may struggle to generalize effectively to new data, resulting in poor calibration even if it appears well-calibrated on the training set. To mitigate this, data is typically divided into training, development, and test sets. The model is trained on the training set, calibrated on the development set, and finally evaluated on the test set. Poor calibration may also arise if the test data originates from a different distribution than the training data, causing the model's predictions to deviate. In both cases, the model may perform well on the training data but fail to generalize accurately to unseen data. Furthermore, inadequate calibration can occur if the sample size used for calibration is too small. The accuracy of estimated probabilities plays a critical role in model calibration, and with limited data, it becomes challenging to obtain accurate probability estimates, leading to sub-optimal or weak calibration. Hence, while calibrating output probabilities may improve the trustworthiness of predictions, it does not guarantee optimal model performance on new, unseen data.

Additionally, it is essential to note that statistical calibration provides an average measure of confidence and does not determine the confidence level for a particular instance and its corresponding specific model prediction. This distinction is significant because there may be a need to ascertain the confidence level for individual instances, rather than relying solely on the average confidence across a dataset of interest. Taking into account "instance-specific confidence" may be of very relevant importance as it allows us to assess the performance of a model on individual examples. That is, unlike statistical calibration, which provides an average measure of confidence, instance-specific confidence offers a more precise evaluation of the confidence level for a particular instance. By comparing model confidence with statistical confidence, we can uncover patterns or clusters within the data. For instance, if the

model consistently underestimates or overestimates the expected statistical confidence for certain groups or examples, it may suggest an irregularity in the model (e.g. a bias towards those groups) or indicate potential anomalies and outliers.

Traditional evaluation measures such as accuracy, precision, and recall provide valuable insights into a model's predictive capabilities by assessing its accuracy in classifying instances into appropriate categories. However, as motivated earlier, achieving good overall performance is often insufficient. It is crucial to also identify cases where the model is prone to making inaccurate predictions. In particular, classification can fail when dealing with outlier instances that originate from a different distribution compared to the training data, known as out-of-distribution (OOD) samples. Although model calibration adjusts the output probabilities to align with the classification accuracy, calibration also assumes that new samples originate from the same distribution as the calibration set (the development set); which is assumed to but could be different and diverging from the training and test set. It is important to note that both accuracy and model calibration, despite being related, represent distinct concepts. It is possible to achieve high (test) accuracy but low calibration, or vice versa. E.g., consider a classifier that randomly assigns predictions relative to the class frequencies in the data. Although this classifier may be well-calibrated, its performance on new and previously unseen data will be poor and randomly biased, resulting in lower accuracy. To effectively identify OOD samples, it is generally necessary to have a model that performs sufficiently well and accurately; otherwise, it can be challenging to detect OOD samples if the model has not even properly learned the true training distribution. In simpler terms, if a model has not learned the underlying real patterns of the data it was trained on, it will be hard to correctly recognize instances that deviate from those patterns and that do not.

State-of-the-art ML models perform well on train and test data but assume that real-world data is from the same distribution, i.e. that the data used for training and testing represent the complete universe of possible examples. This assumption, known as the *closed-world assumption* (Prince, 2022; Yang et al., 2022; Fei and Liu, 2016; Parmar et al., 2023), poses a challenge and is not met when encountering OOD data, which are therefore atypical samples. One fundamental assumption in OOD detection is the unavailability of large volumes of such atypical samples. If such samples were accessible, then supervised learning could be simply employed to classify them as typical or atypical instead. Thus, per definition, direct training on OOD examples is not feasible. Otherwise, training on such OOD samples would invalidate their "out-of-distribution" status because incorporating them into the training set would render them "in-distribution", thereby contradicting the initial and fundamental definition of being "out-of-distribution".

There are five primary scenarios and reasons for OOD examples (Prince, 2022):

1. *Unexpected inputs*: Stemming from measurement inaccuracies, missing data, erroneously included information, or data that does not belong to the dataset, these inputs deviate from the norm. The data here itself is not typical.
2. *Defects*: Most of the data sample is typical, but a localized portion exhibits an inconsistency or anomaly. The data sample as a whole is typical and belongs in the dataset, but there is a specific part of it that has a problem or defect.
3. *Distribution shift*: Occurs when the observed data's distribution changes over time. It happens when (at some point) the model encounters data in real-world use that significantly differs from that it was trained on.

4. *Statistical outliers*: Atypical examples might arise if the initial distribution displays heavy tails. Though not strictly OOD examples, effectively managing these instances can pose a challenge for an ML model.
5. *Malicious activity*: Deliberate manipulation of the ML system's output by an adversary through the introduction of unusual and targeted inputs; such as adversarial attacks.

The concept of OOD detection encompasses several closely related tasks, such as outlier detection, anomaly detection, novelty detection, multi-class OOD detection, and open-set recognition. However, these terms are sometimes used interchangeably or inconsistently in the literature, so caution is advised when referring to them. A comprehensive overview and big-picture perspective are provided by Prince, 2022 (for a visual overview of the hierarchical terminology see Prince, 2022). Below is a concise summary of the key distinctions:

1. **Outlier detection** involves identifying and eliminating infrequent or improbable instances in an unlabeled dataset, often done before actually applying the data to downstream tasks or training the model.
2. **Anomaly detection** involves determining if a new example belongs to the same distribution as an unlabelled dataset of in-distribution examples.
3. **Novelty detection** involves determining a score for a new example indicating its novelty in regards to an unlabelled initial dataset. It is typically assumed that a novel instance comes from a new region or mode of a changing probability distribution or something not seen during training.
4. **Open-set recognition** is when a labeled dataset with known classes is provided, and the objective is to determine whether a new example belongs to one of these known classes or to an unseen class.
5. **Multi-class OOD detection** utilizes a labeled training set to determine whether a new, unseen example belongs to one of the known classes or if it should be classified as an outlier. It is similar to open-set recognition but focuses on identifying non-typical data that may be corrupted or different in some way and do not (primarily) belong to previously unseen classes.

Yet, often only the following two main scenarios are distinguished: outlier detection and novelty detection (Pimentel et al., 2014). In outlier detection, the training data is contaminated by outliers, and the goal is to model the regions where the majority of the data is concentrated while ignoring the anomalous observations. On the other hand, novelty detection assumes clean training data and aims to determine if a new observation belongs to the same distribution as the training data or not. In this case, an outlier is also referred to as a novelty.

Outlier detection methods can be classified into different categories based on various criteria, such as clustering, classification, neighbor-based, statistical, information-theoretic, and spectral methods (Belhaouari et al., 2021). However, a more general distinction can be made between two types of approaches: embedding-based and prediction-based. Embedding-based approaches transform data into a lower-dimensional space and use the transformed data to identify anomalies. Prediction-based approaches train a model on a dataset and use the model to predict the output of new data points. If the prediction differs significantly from the actual output, the

data point is considered an outlier. The choice of approach for anomaly detection depends on various assumptions, factors, and trade-offs.

These factors include (Belhaouari et al., 2021): (1) Labeled data availability and quality: In prediction-based approaches, labeled data is typically required for training models through supervised or semi-supervised learning. Conversely, embedding-based approaches can utilize unlabeled data through unsupervised or self-supervised learning. Embedding-based approaches are better suited when labeled data is scarce, unreliable, or expensive to obtain. (2) Data complexity and dimensionality: By projecting data into a latent space, embedding-based approaches reduce data dimensionality and complexity. This transformation enables easier differentiation between normal and anomalous data. (3) Interpretability and explainability of results: Prediction-based approaches often yield more interpretable and explainable results by establishing connections between predictions and input features. On the other hand, embedding-based approaches generate more abstract and obscure representations that may be more difficult to comprehend or justify. Therefore, prediction-based approaches may e.g. be interesting and preferred for applications that prioritize interpretability and explainability. (4) Data type and format: Embedding-based approaches can detect outliers in any unlabeled dataset that possesses a meaningful numerical representation. Prediction-based approaches, however, are primarily suitable for labeled prediction datasets (e.g. classification or regression).

Outlier detection is a fundamental problem in ML that has many applications in domains such as fraud detection, anomaly detection, data cleaning, and quality control (Singh and Upadhyaya, 2012; Smiti, 2020; Bansal, Gaur, and Singh, 2016). Again, its main objective is to identify objects that significantly deviate from the majority of objects within a given dataset.

In this thesis work, we present a new method for identifying OOD instances using a prediction-based approach. We build our approach on the premise that while statistical confidence offers a general evaluation of a model's prediction confidence, it does not factor in the variation in the model's calibration across different instances and neglects the degree to which the model's predicted probabilities accurately align with its calibration for a given instance. Our approach directs its attention specifically to this misalignment. Our method involves several steps: First, the probability estimates of a multi-class classification are adjusted by training separate calibration models for each class using a weighted one-vs-rest approach. The inclusion of class weighting is important here because it addresses the inherent imbalance between positive and negative classes in one-vs-rest models. These calibration models are used to transform the original probabilities into calibrated probabilities. We then measure the difference in misalignment between the calibration of each class and the sum of the complement calibrations. The idea is that atypical and abnormal instances or outliers will exhibit a greater misalignment, as they are also OOD for the calibration process. Our approach takes advantage of the differences in the binary one-vs-rest model calibrations among all classes. We utilize the norm of the discrepancy vector, which we term "surprisal," as a measure of confidence estimation in the output predictions. This ultimately aids in the recognition of OOD instances and improves novelty detection.

In a broader context, we distinguish between two fundamental concepts: "reliability" and "certainty," which represent varying degrees and types of discrepancies. "Reliability" pertains to the statistical accuracy confidence of a model, reflecting how well the model's average performance aligns with actual probabilities. Calibration, for example, is therefore in general employed to enhance the average reliability of

predictive outputs, ensuring they better correspond to the true expected probabilities. Conversely, "certainty" denotes the calibration confidence of the model for a specific prediction or instance. It represents the level of confidence we hold regarding the output for a given and specific input. For instance, a model might exhibit high overall reliability, yet display significant uncertainty about a particular input. Importantly, this certainty is distinct from the output probability of a class (i.e. the softmax score). An overly confident output, for instance, could even raise suspicion, especially in cases of OOD predictions.

By making this distinction, it becomes evident that we seek both a *reliable* overall model and *certain* input-specific predictions. In fact, if the model lacks reliability, certainty about individual predictions also diminishes. The concept of "surprisal," as we will define it, measures the extent of disparities between reliability and certainty across all class predictions, i.e. output predictions. This serves as an indicator of "anomaly," which we will leverage for the detection and identification of outliers, i.e. instances that deviate from the established patterns of the model; thereby assisting in the detection of OOD instances.

We will demonstrate that depending on the magnitude of this error vector, diverse interpretations and measurements arise. For instance, a measure of "novelty" can emerge and be integrated, indicating and considering the degree of unexpectedness; with lower values suggesting higher reliability, and higher values indicating greater anomalies. This is tied to a level of "confidence" which is determined and affected by the vector norm (i.e., magnitude), providing insights into the degree of model certainty during prediction (for a specific object). Therefore, the most severe anomalies are regarded as those with high surprise scores, which correspond to those with high novelty and low confidence scores.

### Take-Away

Calibrating Machine Learning models is crucial for accurate predictions, but many neural network models lack calibration despite high accuracy. Detecting out-of-distribution (OOD) samples is challenging. Various OOD detection tasks exist, and this work proposes a new method using class-specific calibration models and a "surprisal" metric to improve novelty detection by identifying outlier instances. Surprisal identifies OOD anomalies by examining differences in model certainty, assuming a valid reliable model is provided, and leveraging concepts like "novelty" and "confidence." It identifies serious anomalies through high novelty and low confidence scores.

### Keywords

Calibration • Out-of-Distribution Detection • Novelty Detection • Anomalies

## 4.12 Manifold Data Representation and Clustering

The rise of Big Data has repeatedly emphasized the need and significance of clustering and outlier detection, essential tasks in both data analysis and ML (Estivill-Castro, 2002). Clustering uncovers patterns and relationships in datasets by grouping similar data points according to their similarity. Meanwhile, outlier detection identifies data points that deviate significantly from the rest, facilitating the detection of anomalies and trend identification. In the context of DNNs, clustering,

and outlier detection are accomplished by employing techniques like centroid-based clustering, hierarchical clustering, density-based clustering, numerous different statistical methods, distance-based methods, and density-based methods on the learned representations of the DNN. However, assessing and comprehending the quality and effectiveness of clustering and outlier detection is difficult because the definition of a "cluster" is often vague and simplistic. This challenge is e.g. also further exacerbated by the uncertainty surrounding the number of actual clusters present.

Besides tedious human expert evaluation and indirect evaluation techniques that assess the usefulness of the method for its intended purpose (Feldman, Sanger, et al., 2007), the evaluation of clustering and outlier detection can be achieved, as already mentioned in previous sections, through formal intrinsic and extrinsic metric assessments. To recap briefly: Intrinsic evaluation employs a cluster quality score, whereas extrinsic evaluation compares the output of clustering to ground truth labels. While extrinsic evaluation is often preferred, it is not always feasible since ground truth labels may not always be (easily or fully) available (Amigó et al., 2009). Intrinsic evaluation methods include statistical analysis, such as setting the threshold for outlier detection based on empirical rules, e.g., three standard deviations below and above the mean (Pukelsheim, 1994). Lastly, clustering involves dividing data into groups such that items in the same cluster are more similar or related to in some ways than those in other clusters (Bramer, 2007; Rokach and Maimon, 2005).

Latent manifolds are lower-dimensional subspaces of data embedded within higher-dimensional spaces that are often used to analyze and identify the structure underlying the data (Tenenbaum, De Silva, and Langford, 2000). Manifold clustering is a technique based on the assumption that data is distributed along a manifold of much lower dimension than the input space (Souvenir and Pless, 2005). However, manifold clustering is challenging because manifolds can have arbitrary dimensions, curvature, and shape, and can be very close to one another (Chen, Lv, and Zhang, 2017). Overall, it aims to partition the data into clusters corresponding to different manifolds present in (and learned from) the dataset.

The Taylor expansion (Thomas, 1992) is given by an infinite series of terms that are expressed in relation to the derivatives of a function at a single point. Generally, for most common functions, the function and the sum of its Taylor series are nearly identical near that point. The Taylor approximation is, thus, a mathematical technique that utilizes polynomials to approximate functions. Taylor's theorem asserts that any smooth function can be closely approximated around a specified point using a polynomial termed its Taylor polynomial. Specifically, the  $k$ -th order Taylor polynomial approximates a function, differentiable  $k$  times, around a given point by employing a polynomial of degree  $k$ .

In this thesis, we employ neural Taylor approximations to tackle the problem of manifold clustering, e.g. to use for outlier detection. We utilize Taylor's theorem, which states that a smooth function can be approximated by a polynomial near a specific point, to approximate the manifold using polynomial functions. This enables us to analyze and identify the data's underlying structure by considering the errors in the Taylor approximation. This is motivated by the expectation, that points residing on the same manifold are likelier to exhibit a smaller manifold approximation error compared to points situated far apart and on different manifolds. An illustration is given in Figures 4.8 & 4.9 below.

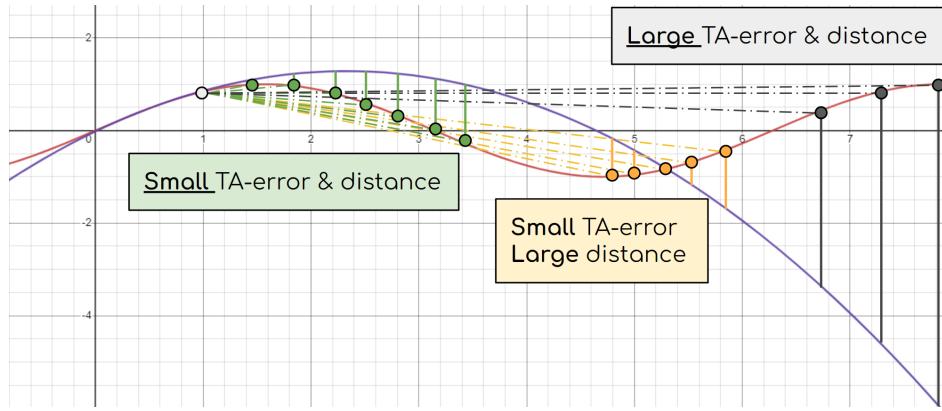


FIGURE 4.8: Second-order Taylor approximation error (TA-error) from a source to other points in a simple 2D space. Points with larger TA-errors, or that are distant from the source, are less likely to *align* with the source. This can be used as a pairwise measure.

### Take-Away

Clustering and outlier detection are key concepts in data analysis that are closely connected. Evaluating their quality poses a challenge, but intrinsic and extrinsic metric assessments can be employed. Manifold clustering is also challenging, due to the arbitrary dimensions, shape, and curvature of latent manifold structures. To address this problem, we present a neural Taylor approximation-based approach for clustering. It involves approximating the manifold using polynomial functions and analyzing the errors in the Taylor approximation to reveal patterns in the underlying structure of the data.

### Keywords

Manifold Clustering • Taylor Expansion • Neural Taylor Approximation

## 4.13 Representative Sampling in Data Streams

Data streams and time series are both concepts in the field of data analysis, but they refer to different things. A data stream is a continuous flow of data that is generated over time, and the data can be infinite. ML on data streams focuses on developing algorithms that can operate on this continuously generated data in real-time. A time series, on the other hand, is a (also a finite or infinite) sequence of data points that are collected at different time intervals, but each data point further has a corresponding timestamp or order index. So, while every time series can be seen as a data stream due to the sequential arrival of data points over time, it's important to note that not all data streams are time series. A data stream, however, transitions into a time series when its data points are organized and indexed based on a specific time order. Also, another slight distinction lies in how they are managed: data streams are commonly handled in real-time. Thus, all incoming data is typically handled and considered immediately. On the other hand, time series data tends to be stored and examined retrospectively, allowing for the analysis of trends, patterns, anomalies, etc. over specific periods or intervals *after* (part of) the data has been collected.

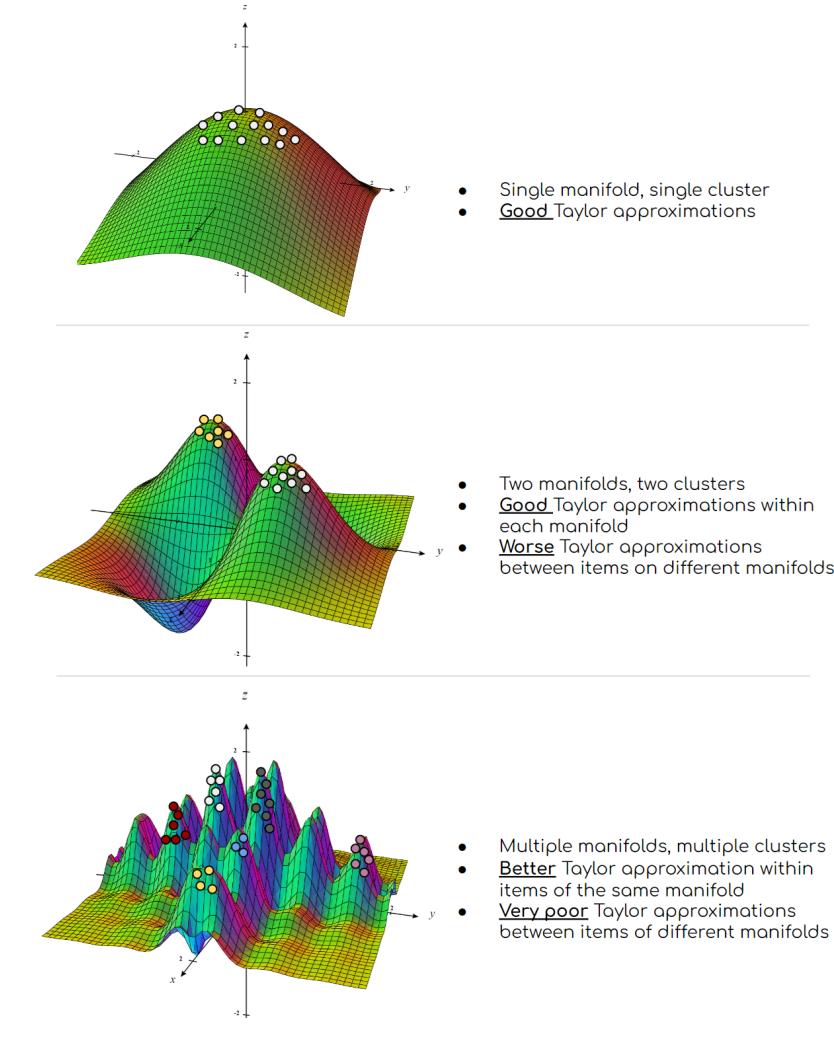


FIGURE 4.9: A manifold clustering illustration, where nearby points on a shared (smooth) manifold yield pairwise good (aligned) Taylor approximations. Proximity and locally good approximations indicate clusters. Proximity in Euclidean space differs from proximity in the manifold space. Same-colored points share the same local manifold.

Analyzing data streams (and hence time series) is an important topic with wide-ranging applications in many areas. Unsurprisingly, the success of ML has therefore also generated increased interest in its application to data streams. However, data streams have unique characteristics, posing novel and non-trivial challenges for ML. First, data streams are very large, potentially infinite, and continuously evolving; and new data points are generated in real-time, unlike in traditional ML datasets where the data is typically static. This makes many methods such as clustering, outlier detection, etc. notably harder since patterns in the data can now also evolve and, hence, change (e.g. two unconnected clusters can suddenly be connected; densities in the clusters may change, etc.). Here, the phenomenon of concept drift is prevalent. Concept drift refers to the situation where the statistical properties of the data change over time. This gradual or sudden transformation renders previously trained ML models progressively obsolete and ineffective. Continuous monitoring, adaptation, and retraining of the ML models may, hence, be necessary to accommodate shifting data distributions. This is important because such changes can render the

previously identified patterns, rules, associations, and assumptions of the learned models invalid; consequently undermining their generalizability.

ML for data streams presents challenges. Streams may be gathered from diverse sources and, thus, arrive at different frequencies; and their arrival rates may unpredictably change or fluctuate. Adapting ML to account for this potential variability is important. Also, unlike traditional datasets with fixed and static data points and features, data streams typically (as already mentioned) possess an unknown and potentially infinite length. Moreover, since they originate from various sources, the data gathered may show differences in data quality, feature distribution, and feature correlation. Such disparities in data result in additional difficulties in applying ML models. In particular, different sources may experience different types and amounts of noise; in which case addressing noise reduction emerges as a crucial concept.

Memory constraints represent primary concerns for (infinite) data streams. Since data streams are continuous and potentially unbound in length (or often extremely large), it is not possible to store all the data points in memory for training the ML models; especially when many of such data streams exist simultaneously (e.g. trajectory streams for different entities). Therefore, the ML algorithms must be designed to work with memory constraints, while still maintaining high accuracy; which can often result in a trade-off. Furthermore, data streams often contain irrelevant or contaminated data records. Standard methods of data cleaning and preprocessing may not work well because the data keeps coming in continuously. Hence, ML algorithms must be robust and able to filter out irrelevant data points in real-time. Lastly, the concept of "order" is crucial; i.e. the order in which data arrives can significantly influence the meaning and performance of the ML models. ML algorithms must take into account the temporal ordering of the data points and adapt to changes in the underlying distribution over time. This requires specialized algorithms such as online learning and online clustering. These characteristics make ML on data streams a challenging task that requires the development of specialized algorithms and methods capable of handling such continuous streams of data efficiently.

In this thesis, we propose an online sampling method based on Reservoir Sampling for maintaining a representative data stream sample that does not rely on prior knowledge of the data stream's length and can handle infinite streams. Our method extends to maintaining a representative sample across different quantiles of the data distribution. Overall, it ensures that each new incoming element in the stream has an equal uniform chance at any time of being sampled despite having an initially unknown or infinite total length; and holds interesting mathematical properties. These properties establish the criteria for defining a "representative sample".

### Take-Away

Challenges in data streams include variable input length, concept drift, varying data arrival rates, and memory limitations. Specialized algorithms are needed to address these issues. The Reservoir Sampling family stands out by effectively handling infinite streams without prior length knowledge, ensuring certain mathematical guarantees.

### Keywords

Data Streams • Time Series • Real-time Data Analysis • Reservoir Sampling

## Chapter 5

# Preceding Project Endeavors

**Disclaimer:** Exact details on the preceding project are governed by a non-disclosure agreement imposed by the BMW Group. Consequently, this section has been written in compliance with the terms of this agreement.

Before this dissertation, an initial effort was made to employ ML to address interests in identifying error patterns within the context of a preceding Master thesis project (agreement, n.d.). However, due to the brevity of the said thesis and limitations in the methods used, satisfactory and reliable results were not achieved. Nonetheless, the potential of utilizing ML in this context was clearly demonstrated and subsequently paved the way for this doctoral thesis.

Due to the imposed non-disclosure agreement, the subsequent sections of this chapter will provide only a theoretical and abstract overview of the approach that inspired and led to this thesis, along with a discussion of key shortcomings in previous attempts and approaches. Some of the underlying reasons for these weaknesses will also be explored, and some suboptimal or unsuccessful choices related to model architecture and encoding representations will be explained. Implementation details and past results are, thus, *not* included in compliance with this agreement. Please note that this section should not be interpreted as a critique of the quality of the previous work, but rather its effectiveness. The ultimate goal is to highlight some key limitations of previous methods to better motivate and understand the reasons and necessity of the novel techniques that will be presented in this thesis. In fact, to better understand the context of this doctoral thesis, it is important and key to be aware of "why" previous methods fell short of expectations and how this knowledge shapes our future approach.

### 5.1 Overview

Figure 5.1 shows the overall structure of an ML process pipeline. The pipeline system is divided into two parallel tracks where one track represents *explanatory*- and the other for *predictive* modeling. The explanatory track is meant for human interpretability, whereas the predictive track focuses on predicting and determining the defect patterns, which are then passed back to the explanatory track to serve as class labels for rule learning. Due to the inherent trade-off between accuracy and interpretability, a straightforward rule-based system might sacrifice generalizability and effectiveness, whereas a more intricate system, although more powerful, might lack transparency. This trade-off is an inevitable aspect to consider.

Briefly summarized, the objective was to leverage AI techniques to derive interpretable rules and conduct an automated search for generalizable patterns within data. This particularly involved generating easy-to-understand decision rules relative to a model's output predictions; i.e. the identified patterns. Additionally, a

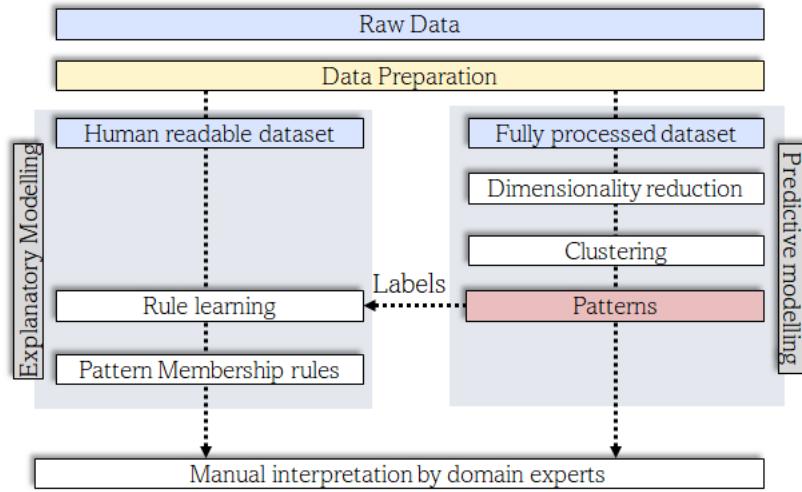


FIGURE 5.1: Overview of the pipeline proposed in (agreement, n.d.).

central objective was to understand and interpret data classifications even when ground-truth labels were absent or imperfect. This was approached using different ML models and methods; aimed to capture intricate patterns automatically, robustly, and reliably. The goal was also to mitigate the negative impact of (potentially) low-quality data labels and handle an unreliable set of observable features. Data was given in multiple formats, such as text, complex structured sensor data, etc. This alone already posed a serious challenge: handling complex, unconventional multi-modal and heterogeneous data.

## 5.2 Data Encoding

Data cannot simply be "fed" into an ML model; it has to first be *encoded*. The type of encoding is dependent upon the underlying structure and format of the data. In our case, the input data does not come in a conventional tabular form that is easily vectorizable (due to the multiple 1:n relations within the database); see Chapter 4.1 for details (and examples) on the hierarchical structure. In particular, complexities arise due to multi-modality, such as data that contains e.g. numerical, categorical, textual, and timestamp features. Data encoding and pre-processing play a crucial role, with poor choices of coding strategies quickly resulting in insufficient model effectiveness, poor performance, etc. In this section, we will now present how this was (naively) approached before the start of this work.

### 5.2.1 Categorical Univalent Features

Given categorical features, the approach used previously was to encode these categories with one-hot-coded vectors, which are particularly easy to understand, implement, and integrate, and are good for small underlying problems with a rather limited and small number of categories. However, this simple and trivial coding approach - and in line with the "no-free-lunch" theorem - has several disadvantages.

The dimensionality of the encoding vectors corresponds to the total number of different categories and therefore scales linearly in each case. Suppose we have a total of  $k$  different categories (e.g. vehicle parts, etc.) for a particular characteristic. An obvious consideration would be the question of whether it is really necessary to

have sparse vectors of size  $k$  that are zero everywhere except in one place. In general, one-hot vector encodings encounter many challenges like high dimensionality and so the susceptibility to the curse of dimensionality, the inability to capture or reflect semantic relations, increased sparsity, suboptimal handling of ordinal variables, and struggles with accommodating new categories (such as when a category emerges in the test set that wasn't in the training data).

Instead, encoding the information into a smaller, stable, compact, and contextual embedding vector would be more practical. This approach not only diminishes dimensionality and complexity (thereby reducing and improving training time) but could also enhance model quality and effectiveness, e.g. in tasks like clustering. This improvement can be particularly relevant if these dense vector representations further capture distinct (semantic) meanings or logical relationships.

### 5.2.2 Tabular Data

The approach used in the master thesis was *cardinality linearization* (CardL), i.e. brute-forcing all possible combinations with respective entities into one very large vector. CardL adds path-dependent dummy columns to create a wide-format tabular data encoding on an arbitrary recursive chain of entities (Figure 5.2).

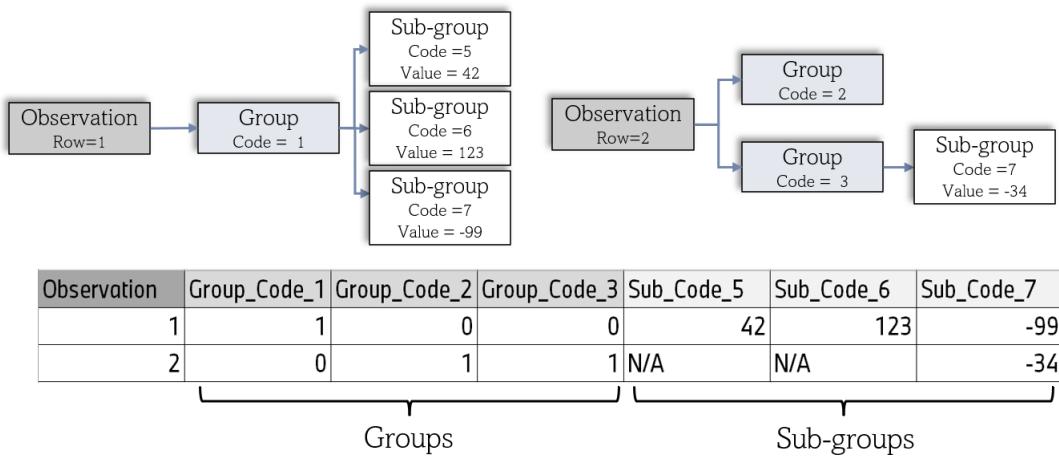


FIGURE 5.2: Example of cardinality linearization as done previously.

However, such a procedure also has decisive disadvantages. First, the dimensionality of the vectors explodes exponentially with the number of relational connections. For example, suppose we have  $k$  groups with  $n_i$  categories of coding sizes and features of size  $m_i$  per group. Then the total number of combinations is  $\prod_{i=1}^{k-1} n_i m_i$ , which can quickly scale up to unacceptable magnitudes.

Secondly, this approach results in unfilled feature values for features that only appear in specific categories (see red box in Figure 5.2). The application of data imputation to counteract this issue is not a sensible strategy as the respective parts would be very sparse and distort the data considerably, which would consequently have an effect on the ML models.

Additional concerns involve the scalability of computations, declining model performance, reduced interpretability (caused by sparsity), and heightened maintenance complexity, rendering it less practical and inefficient for use.

So overall, by aiming to consolidate relational connections and features into a single expansive vector, CardL presents significant drawbacks that could be mitigated and avoided with an improved encoding structure.

### 5.2.3 Text Data

At first, text data was processed using tedious and time-intensive manual regex rules and similar methods, which fell significantly short of being optimal, time-efficient, and scalable<sup>1</sup>. Text data was then later encoded using elementary pre-trained models such as BERT in a somewhat simplistic manner. While groundbreaking upon BERT's release, this approach may now be considered a naive solution or baseline approach and lacks the effectiveness required in many of today's complex applications. Particularly, it lacks the incorporation of use-case-specific architectures and derived loss functions; especially to account for use-case-specific challenges and limitations. Fine-tuning and domain adaptation are often necessary.

Thus, although employing pre-trained BERT models for clustering and embedding generation is very common and has proven to show great results, recent advancements in specialized models, architectures, (unsupervised) learning techniques, etc. have emerged; including larger and stronger language models. For instance, advanced techniques like augmented SBERT have demonstrated improved results in encoding and utilizing semantic meaning and context within sentences. This can enhance performance across a range of NLP tasks, such as measuring sentence similarity, conducting semantic searches, question answering, etc.

Also, given that the text data can be presented in various languages, the use of multilingual language models here may be interesting. This differs from alternative approaches like disregarding non-English data or automatically translating them into English. In fact, translating texts into English requires caution as it can readily result in inaccuracies and potential loss or corruption of information, thereby introducing additional uncertainty and complexity to the pipeline.

## 5.3 Dimensionality Reduction

To reduce dimensionality, the CardL-encoded tabular data underwent processing via a *Stacked Denoising Auto-Encoder* (Vincent et al., 2010). The intention was to attain a more concise and better representation while preserving meaningful semantic information. Denoising Auto-Encoders (AE), variations of conventional AEs, aim to mitigate the risk of learning mere identity functions by intentionally corrupting input signals (e.g., using noise), forcing the model to identify and eliminate this noise during learning.

However, several crucial considerations arise with this method, particularly concerning the nature of the data and its encoding. The concept of denoising becomes challenging when handling sparse input vectors dominated by zero entries (as in one-hot encodings and CardL); e.g. because randomly deactivating vector components (setting them to zero) has minimal impact, etc.

Another criticism is the unnecessarily large input dimensionality from the CardL and the complexity introduced into the AE, which can result in having to use much larger models to effectively handle this, which thus increases training time, lowers prediction quality, and increases prediction error. In fact, if decoding is not done

---

<sup>1</sup>We have clearly outlined and opposed the utilization of rule-based (regex) systems for NLP use cases in Chapter 2.2.6 and provided examples.

accurately and crucial factors are misunderstood, it can lead to unintended consequences. Also, an overly small embedding dimension or bottleneck layer in AEs can limit the models' ability to represent complex data; in particular, if the bottleneck is too strong given such large input data (due to CardL). This can lead to information loss, reduced expressiveness, lower performance, etc. It's crucial to strike a balance between dimensionality reduction for efficiency and maintaining enough information for the model to learn effectively. In particular, when the embedding dimension is insufficient, the model may struggle to learn and represent the complex relationships and patterns present in the data; as a result, it may not generalize well; that is, a small embedding dimension can frequently lead to underfitting rather than overfitting. Similarly, an excessively large embedding dimension may contribute to overfitting by allowing the model to memorize noise in training data. With the CardL encoding, the primary concern lies in having embedding dimensions that are too small, leading to an overly abrupt bottleneck; and so, a performance loss.

Ultimately, CardL + AE also does not directly consider the essential hierarchical and relational aspects of the data, failing to fully and effectively utilize its valuable structural and relational information.

## 5.4 Error-Pattern Inference

Identification of error patterns was conducted via both: ordinal clustering on the input features and *deep clustering* on the deep representations, i.e. clustering on the raw input encodings or on the embedding layer of the AE. Clustering on inputs has the disadvantage that it does not actually "learn" patterns, but instead simply groups similar input representations that are close together in the space. However, semantically similar inputs may be dissimilar in the input space.

Deep clustering is much better in that it clusters on "learned" embedded representations. Yet, when performing deep clustering on self-supervised AEs, correlations, and patterns regarding labels are not considered or learned. Although one might argue that these labels might be incomplete and incorrect and should thus not be considered during learning, one could equally argue that leaving them out could be even worse as we would disregard learning from implicit important information that would otherwise be very hard to learn or extract from solely the model's inputs.

### Take-Away

A preceding Machine Learning project aimed to detect error patterns in complex hierarchical data is discussed, which faced limitations due to methodology constraints and a tight timeline. Challenges included issues with data encoding, particularly the use of one-hot encoding for categorical features and cardinality linearization on complex database structures, leading to high-dimensional sparse vectors, and more. Methods like Stacked Denoising Auto-Encoders for dimensionality reduction struggled with sparse input vectors and did not fully leverage hierarchical structures. Naive error pattern inference methods such as ordinal clustering and deep clustering proved ineffective and overlooked label correlations, disregarding valuable information.

### Keywords

Preceding Approaches • Flaws • Encoding • Dimensionality • Error Patterns



## Chapter 6

# Related Work

### 6.1 Neural Networks Based on Complex Numbers

**Notice:** Our work and findings on complex-valued NNs (presented in this thesis) have been published in Guimerà Cuevas, Phan, and Schmid, 2023. It was inspired and builds upon our previous research (Guimerà Cuevas and Phan, 2021), but differs significantly as it has been substantially improved and modified (both the complex architecture and its mathematical framework); thus, constitutes novel contributions. In particular, this novel contribution has been accepted for publication, after peer review. The Version of Record is available online at: [https://link.springer.com/chapter/10.1007/978-3-031-33374-3\\_28](https://link.springer.com/chapter/10.1007/978-3-031-33374-3_28). Use of this Accepted Version is subject to the publisher's Manuscript terms of use. Therefore, please refer to Guimerà Cuevas, Phan, and Schmid, 2023 for specific details.

Complex-Valued Neural Networks (CVNNs) have been explored since the 1990s (Bassey, Qian, and Li, 2021; Little, Gustafson, and Senn, 1990; Hirose, 1992; Clarke, 1990), but their development lagged behind real-valued (RV) networks due to the complexity of complex-valued (CV) back-propagation (Liu et al., 2017). Modern frameworks have mitigated these challenges, but designing CVNNs still presents difficulties, particularly in mapping real numbers to complex ones and defining suitable activation functions. Effective learning requires the network to be holomorphic, satisfying the Cauchy-Riemann (CR) equations (Ahlfors, 1953), yet most loss functions of interest are RV or non-holomorphic.

CVNN architectures are often defined by their activation functions, with non-holomorphic functions categorized as split activations: Type A (bounded real and imaginary parts) or Type B (bounded magnitude) (Kuroe, Yoshida, and Mori, 2003). While CVNNs typically handle CV variables and parameters (Hirose, 2003; Hirose, 2012), some only require CV weights. Early CV back-propagation methods emerged in the 1990s (Leung and Haykin, 1991; Benvenuto and Piazza, 1992), and MVNs for complex numbers were introduced in 1992 (Aizenberg and Aizenberg, 1992), evolving to incorporate advanced back-propagation and periodic activation functions (Aizenberg, Moraga, and Paliy, 2005; Aizenberg, Paliy, and Astola, 2006; Aizenberg et al., 2016; Lupea, 2012; Aizenberg and Moraga, 2007; Aizenberg, 2011b).

A Type A activation function  $f_C(z) = f_{\mathbb{R}}(x) + if_{\mathbb{R}}(y)$ , using the sigmoid function, was shown to improve generalization by forming orthogonal decision boundaries (Nitta, 2004). Additionally, RV inputs have been mapped to complex numbers for use in CVNNs (Amin and Murase, 2009). CVNNs have also been adapted for convolutional layers, including techniques for batch normalization and weight initialization (Trabelsi et al., 2017).

## 6.2 Heterogeneous Graph Learning

As we will demonstrate later, the task of applying DL to HDBs can be reduced to a heterogeneous graph-learning (HGL) problem. This chapter aims to present a basic summary and overview of pertinent research on the topic of HGL. A full and extensive summary of existing literature is given by Wang et al., 2022a.

GNNs are suitable when data is obtained from non-Euclidean domains and represented as graphs with complex relationships (Wu et al., 2020). However, traditional and commonly used GNNs are designed for homogeneous graphs, in which all nodes have the same features, and all relations indicate the same type of relation. Hierarchical GNNs (Sobolevsky, 2021) aim to unify conventional neural networks with GNNs by connecting layers both externally and internally, but they still struggle with processing nodes that have different database features, even with a hierarchical graph-like model design. Heterogeneous graphs (HG) are graphs with structural interconnections (edges) that link nodes of multiple types; with each node containing unstructured or structured information. That is, HGs contain data with various entities and diverse relationships. The goal is to obtain a meaningful vector representation for each node or graph, which allows for further downstream applications (Zhang et al., 2019). This is difficult not just because of the diversity of nodes and edges, but also because of the heterogeneity of feature attributes and content. In particular, applying standard GNN message-passing to HGs is not straightforward since the node and edge features of different types can (obviously) not be processed by the same functions owing to variations in feature type and dimension. The feature aggregation in conventional message-passing GNNs, however, is based on an *aggregation* function that assumes all nodes have the same features and feature types (in particular, equal feature dimensions). Therefore, two main challenges of HGs are (Wang et al., 2022a): (1) The complex structure caused by different types of nodes and edges; (2) The heterogeneity of feature attributes.

A selection of typical methods that use both graph structure and feature attributes to learn low-dimensional node representations for HGs is given below (see the comprehensive survey by Wang et al., 2022a):

- **GCN** (Schlichtkrull et al., 2018): Graph Convolutional Networks update node representations by passing messages between neighboring nodes in a graph. By stacking multiple GCN layers, the network learns increasingly complex representations of the graph, from capturing simple local connectivity patterns in the first layer to more global patterns that span longer distances in deeper layers. The hierarchical representations learned by GCNs can improve performance on downstream tasks that require capturing complex dependencies between nodes in the graph. To handle heterogeneity, GCNs use feature vectors associated with each node and edge that encode their type and attributes. These vectors are transformed by learnable weight matrices before being propagated through the message-passing scheme.
- **HetGNN** (Zhang et al., 2019): A **Heterogeneous GNN** model that uses separate embedding layers for each feature type and using attention mechanisms to dynamically weigh the importance of different feature types during message passing.
- **MAGNN** (Fu et al., 2020): Employs a combination of **Meta-path-based** and **Attention-based** aggregation with **GNNs** to capture the complex relationships between nodes and edges of different types. It learns node embeddings by

computing embeddings for each node along the specified meta-paths and then aggregating them using an attention-based weighted sum. Finally, the latent vectors obtained from multiple meta-paths are fused to produce the final node embeddings.

- **HetSANN** (Hong et al., 2020): A **Heterogeneous Structural Attention Neural Network** for HGs that encodes the structural information of HGs without relying on meta-paths. It achieves informative representations by employing an HG structural attention mechanism, which directly attends to the neighboring nodes and edges to generate node embeddings. This allows the model to capture the local and global structural information of the HGs and enables it to learn embeddings for downstream tasks.
- **HGT** (Hu et al., 2020): A **Heterogeneous Graph Transformer** architecture that draws inspiration from the Transformer model used in natural language processing. HGT thus also employs attention mechanisms to learn node and edge representations, which allows it to selectively focus on different parts of the graph during both training and inference.
- **GATNE** (Cen et al., 2019): A **Graph ATTention NEtwork** that employs a skip-gram model to optimize the embeddings obtained. It involves predicting the context nodes of a target node based on its embedding.
- **GTN** (Yun et al., 2019): A **Graph Transformer Network** that uses multiple Graph-Transformer layers to perform message passing and feature updating operations on the nodes and edges of the graph, and attention mechanisms to focus on important parts of the graph. By identifying meaningful links between nodes that were not originally connected, GTNs can generate novel graph structures.
- **RSHN** (Zhu et al., 2019): A **Relation Structure-aware Heterogeneous GNN** that embeds both nodes and edges in HGs without prior knowledge such as meta-paths. It integrates a Coarsened Line GNN to reveal edge-centric relation structural features that respect the latent associations of different types of edges based on the coarsened line graph, followed by a Heterogeneous GNN to leverage implicit messages from neighbor nodes and edges propagating among nodes in heterogeneous graphs.

Overall, multiple approaches have been explored for encoding knowledge in relational DBs. Further intents involve complete brute-force denormalizing of the DB by materializing the full join of all tables or representing it as a knowledge graph (KG) (Arora and Bedathur, 2020). Another approach is to use word embeddings to enable semantic queries in relational databases in an unsupervised manner, with vectors that encode contextual relationships (Bordawekar and Shmueli, 2017). Additionally, symbolic representations can be encoded into a continuous vector space by learning embeddings for entities and operators for relations (Bordes et al., 2011). Efficient, easy-to-train, and scalable holographic embeddings using circular correlations have also been proposed for creating vector space representations of entire knowledge bases or graphs, and for link prediction (Nickel, Rosasco, and Poggio, 2016). For directed graphs, another successful method for link prediction denotes relationships by translations operating on low-dimensional embeddings of entities (Bordes et al., 2013). Relational ML techniques for KGs are discussed in several review papers, including Nickel et al., 2015; Xu et al., 2018; Cai, Zheng, and Chang,

2018. It is important to note that although KGs and DBs are often used interchangeably, the key distinction is that KGs focus on encoding semantic relations and concepts, while DBs store actual data facts (Brodie and Mylopoulos, 1986).

This work aims to address the challenges of DL for HDBs by directly exploiting the hierarchical structure and conducting uni-directional message-passing from the bottom of the HDB tree upwards. This involves recursively aggregating sub-node representations and mapping them to embeddings of fixed predefined sizes, with the option to consider entity-wise importance weights. Unlike heterogeneous GNNs, this approach leverages the topological information of the hierarchical data, taking into account that inaccuracies or noise on higher-level nodes may have a greater impact than those on leaf nodes. This is achieved by (optionally) specifically ensuring that information is not lost, or at least information loss is minimized, at each hierarchy level; and can be prioritized (i.e. weighted) accordingly. In addition, not all HDB tables may be equally relevant or informative. Thus, this work also introduces a way to better ensure optimal relevance weighting (i.e. consideration) of entities given a specific objective, e.g. classification. Each objective loss can also further directly be optimized at each individual level of the hierarchy; not only at the highest level (i.e. the root).

### 6.3 Robust Feature Normalization

**Please note:** Our contribution on "Adaptive Tanh-Normalization" has been accepted for publication, after peer review in Guimerà Cuevas and Schmid, 2024b. The Version of Record is available online.

*Tanh*-normalization has been shown to be effective on multiple tasks (Bhanja and Das, 2018; Jain, Nandakumar, and Ross, 2005; Ribaric and Fratric, 2005). It counts as being very robust to noise but is more intricate to compute than other simple methods (e.g. z-scores). TE was originally (Hampel et al., 2011) defined as:

$$\phi(x) := \frac{1}{2} \left[ \tanh \left( 0.01 \frac{(x - \mu_\Psi)}{\sigma_\Psi} \right) + 1 \right] \quad (6.1)$$

$\mu_\Psi, \sigma_\Psi$  are the mean and standard deviation of the *Hampel* estimator (HE) scores (Hampel, 1974; Shevlyakov, Morgenthaler, and Shurygin, 2008). The HE is a three-part re-descending M-estimator (Hampel, 1973); it can entirely reject gross outliers while not totally dismissing moderate outliers using the re-descending function  $\Psi$ :

$$\Psi(x) = \begin{cases} x, & 0 \leq |x| \leq a \\ a \operatorname{sign}(x), & a \leq |x| \leq b \\ \frac{a(c-|x|)}{c-b} \operatorname{sign}(x), & b \leq |x| \leq c \\ 0, & c \leq |x| \end{cases} \quad (6.2)$$

The parameters  $a, b, c \in \mathbb{R}$  have an effect on points at the distribution tails and determine the robustness of the estimator.<sup>1</sup> Note that  $\Psi$  itself is not the normalization. If the influence of a high number of distribution tail-points is minimized based on the parameters, the estimate becomes more robust to outliers but less efficient (optimal). Here, "efficiency" refers to utilizing data for precise estimates. A less

<sup>1</sup> $\Psi$  functions are non-decreasing near the origin, but decreasing (towards zero) far from the origin. Andrew's Sine Function (Hinkley, 1973) and Tukey's Biweight Function (Beaton and Tukey, 1974) are alternatives to Hampel's  $\Psi$  function. Redescending M-estimators to handle outliers are also used in regression contexts (Khan et al., 2021b).

efficient FS may distort the data more but can be more resilient to noise and variations. Therefore, in contrast, if the estimate is influenced by a large number of tail points, it becomes less robust but more efficient (linear dependencies are better preserved). The tanh-distribution in the transformed domain has a mean of 0.5 and a standard deviation of about 0.01. The spread of the normalized scores is determined by the constant (i.e. 0.01) in the tanh-normalization equation (Jain, Nandakumar, and Ross, 2005). By definition, the range of the tanh-estimator is  $[0, 1]$ , but it can easily be constraint linearly within the range of  $[-1, 1]$  using the normalization of  $\phi'(x) := \tanh(0.01 \frac{(x - \mu_\psi)}{\sigma_\psi})$ .

Choosing the right *Hampel* parameters is crucial but challenging. Thus, a modified tanh-normalization (Latha and Thangasamy, 2011) was proposed that instead used the raw features' mean and standard deviation rather than the *Hampel* scores; considerably simplifying and rewriting the normalization of each feature  $x \in X$  to  $\phi''_X(x) := \tanh(0.01 \cdot \bar{x})$ ; where  $\bar{x}$  is the respective standardized value. The intrinsic complexity associated with the *Hampel* function was so fully removed, resulting in a less complex formula (i.e. simpler computation). They did, however, still employ a default constant value of 0.01 (which will be referred to as the *spread-value* in this paper); and can be seen as a configurable hyper-parameter. Thus, although the normalization is now both robust and fast to compute, it still requires correct spread parameter estimations (Jain, Nandakumar, and Ross, 2005) or manual tweaking; as also noticed by Atrey et al., 2010, where the constant factor in the tanh normalization for the fingerprint modality was increased to 0.1. Further, apart from its use in tanh estimators, other work (Nandakumar et al., 2007) also reported e.g. that face match scores generated using a multi-layer perceptron classifier and a tanh non-linearity function presented problems: the outputs peaked at -1 and 1, resulting poor performance. This shows the importance of properly pre-scaling features before applying the tanh function.

The tanh function was also utilized in different FS approaches. For example, Linear-Tanh-Linear (LTL) (Singh and Gupta, 2007) was presented for biometric systems and is based on the tanh-estimator over a set of scores  $O_k^G$  and a set of imposter scores  $O_k^I$  for a characteristic  $k$ . It maps the non-overlapping region of imposter scores to a constant value of zero and the non-overlapping region of scores to a constant value of one. The tanh-estimator is then used to map the values of the overlapping region between  $O_k^G$  and  $O_k^I$  via a nonlinear tanh function (again with fixed spread-value 0.01). The authors concluded that LTL is also efficient and robust. Furthermore, the tanh-function's resilience to anomalies has also found application in trimmed estimators (Leonowicz, Karvanen, and Shishkin, 2005) for robust averaging when the number of trials is small and the data is highly non-stationary or contains outliers; and also used to compress over-large values on images to decrease their effect on later stages of processing (Tan and Triggs, 2010). Lastly, a similar robust normalization method, yet without the use of tanh, is the double sigmoid normalization (Cappelli, Maio, and Maltoni, 2000), which converts scores within a region linearly and scores outside the region non-linearly. However, it also necessitates cautious tweaking of parameters.

Returning to the implications of selecting the spread value  $\alpha$ , there is no justification for why any given fixed  $\alpha$  should always be optimal for all feature distributions. In essence, it's far more plausible that each feature distribution will have its own (ideal) spread value, and that a spread of e.g. 0.01 may even be counter-productive for training in certain cases. Previous work on tanh-normalization has not sufficiently addressed this concern, and given the impact of feature scaling, a robust and adaptable feature normalizing approach is highly desirable. Therefore,

this work addresses the challenge of calculating the feature-wise ideal spread-value  $\alpha$  by building on previous work (Latha and Thangasamy, 2011) which computes the mean and variance of the features without the *Hampel* estimator and instead utilizes feature-based standardization. Rather than using a predetermined, fixed default spread, this paper will outline and propose a different technique for directly identifying the best  $\alpha$  in  $\tanh(\alpha\bar{x})$ ; and its key importance in terms of training convergence will be highlighted and discussed.

## 6.4 Model Output Calibration

**Please note:** Our work and findings on Calibration Misalignment as a Post-hoc Approach for Out-of-Distribution Detection in Deep Neural Networks (presented in this thesis) have been accepted for publication in conference proceedings (Guimerà Cuevas and Schmid, 2024a).

Model calibration is an important aspect of ML, as it allows to align the output probabilities of a model with the true probabilities of the data; such as given in the real-world. Various methods exist and can be used for model calibration:

A naive calibration method is Histogram Binning (Zadrozny and Elkan, 2001), which involves partitioning the predicted probabilities into a fixed number of bins and estimating the true probabilities by calculating the mean or mode of the predictions in each bin. This approach is simple to implement and can work fast and well for models with already overall "more or less" well-calibrated probabilities. However, it may not be effective for models with complex probability distributions.

Platt scaling (Platt et al., 1999) is another popular method for model calibration that involves fitting a logistic regression model to the predicted probabilities, using the true class labels as the target variable. The output of this calibration model can then be used to adjust the probabilities predicted by the original model. Platt scaling is widely used and has been shown to be effective for a wide range of models. However, logistic calibration is designed to handle class scores that are normally distributed, but this assumption may not hold true for various classifiers; e.g. the logistic curve family does not encompass the identity function; and thus uncalibrate a perfectly calibrated classifier (Kull, Silva Filho, and Flach, 2017). Isotonic regression is a non-parametric method (Zadrozny and Elkan, 2002), which fits a non-decreasing function to the predicted probabilities. Although it is a powerful method, it is prone to overfitting, particularly on smaller datasets (Kull, Silva Filho, and Flach, 2017). Beta calibration (Kull, Silva Filho, and Flach, 2017) was therefore proposed to address these issues by using a calibration map based on the Beta distribution. Temperature scaling (Guo et al., 2017) is also a simple and effective method for calibrating the outputs of a model, by adjusting the logits (i.e. the pre-softmax outputs) by a temperature parameter, and then applying the softmax function to derive the predicted probabilities. The temperature parameter can be tuned to optimize calibration performance and has been shown to be effective in improving the calibration of DNNs. Matrix- and vector scaling (Guo et al., 2017) is a multi-class extension of Platt scaling that involves learning a linear transformation of the logits that maps them onto the calibrated probabilities. This transformation can be represented by a matrix, which is learned from a calibration dataset. Once learned, the matrix can be applied to the logits of new examples to obtain the calibrated probabilities. Vector scaling is a variant of matrix scaling that uses a diagonal weight matrix to reduce the number of parameters. Dirichlet calibration (Kull et al., 2019) is another method that can also handle multi-class classification problems that uses a Dirichlet distribution to model

the relationship between the predicted probabilities and the true probabilities of the classes. It consists of log-transforming the uncalibrated probabilities, applying a linear layer, and then performing a softmax function. Other approaches are Bayesian methods (Gelman et al., 1995), maximum-likelihood calibration (Saerens, Latinne, and Decaestecker, 2002), etc.

Outlier detection is the identification of unusual data within a dataset that deviates from the norm. This can be seen as a one-class classification problem where a model is trained exclusively on normal data and subsequently used to detect abnormal data based on a distance measure and threshold (Pimentel et al., 2014). For novelty detection, various methods exist, including statistical techniques, NNs, support vector machines, and clustering approaches (Markou and Singh, 2003; Pang et al., 2021; Salehi et al., 2021; Ding et al., 2014); usually categorized into probabilistic methods, linear models, proximity-based approaches, outlier ensembles, or graph-based methods. The effectiveness of each method varies depending on the data characteristics and distribution, data assumptions, and application domain. Challenges in novelty detection include handling high-dimensional data, concept drifts, imbalanced data, and noise (Seliya, Abdollah Zadeh, and Khoshgoftaar, 2021; Gruhl, Sick, and Tomforde, 2021; Din et al., 2021; Wang et al., 2023; Wang et al., 2020b).

Overall, these approaches can further be categorized as (Prince, 2022; Boukerche, Zheng, and Alfandi, 2020): one-class classification, probability, and generative models, statistical methods, reconstruction methods, model consistency methods, and auxiliary task methods. These approaches differ in how they define and measure the similarity between new examples and the in-distribution data. One-class classification (Ruff et al., 2018; Salehi et al., 2021) trains a classifier to distinguish between in-distribution and out-of-distribution examples. Probability and generative models (Nalisnick et al., 2018; Eduardo et al., 2020) fit a probability distribution to the in-distribution data and classify examples with low probability as out-of-distribution. Statistical methods (Ren et al., 2019; Morningstar et al., 2021; Wang, Bah, and Hammad, 2019) compare the statistics of the input example to those of the in-distribution data and classify examples with anomalous statistics as out-of-distribution. Reconstruction methods (Abati et al., 2019; Chen et al., 2017) train an autoencoder to encode and decode the in-distribution data and classify examples with poor reconstruction quality as out-of-distribution. Model consistency methods (Xiao et al., 2021; Choi, Jang, and Alemi, 2018) train (multiple) models or ensembles on the in-distribution data and classify examples where the models disagree as out-of-distribution. Auxiliary task methods (Liu et al., 2018; Golan and El-Yaniv, 2018) train a model to perform a secondary task on the in-distribution data and classify examples where the performance on this task is low as out-of-distribution.

Methods can also be divided into ad-hoc and post-hoc OOD methods: *Ad-hoc* methods involve integrating OOD detection into the model training process, where the model not only learns its main task (e.g., classification or regression) but also extends the training to prepare to detect OOD instances. Conversely, *post-hoc* methods analyze the model's output to detect OOD instances without influencing the training or modifying the training architecture; conducting OOD detection afterward.

Examples of ad-hoc methods are (Prince, 2022): OOD detectors for NNs using temperature scaling, noise or decomposition (Liang, Li, and Srikant, 2017; Hsu et al., 2020), the membership loss (Perera and Patel, 2019) (which is used in addition to the standard cross-entropy loss), or other architectural changes on the training, e.g. that integrate an auto-encoder and perform classification on the bottleneck layer.

Post-hoc methods include: OpenMax (Bendale and Boult, 2016; Ge et al., 2017)

which estimates confidence levels in predictions using softmax scores and probability distributions, but also methods that detect outliers based on density e.g. Local Outlier Factor (LOF) (Breunig et al., 2000), Local Correlation Integral (LOCI) (Papadimitriou et al., 2003), Global-Local Outlier Scores from Hierarchies (GLOSH) (Campello et al., 2015), clustering membership strengths (McInnes, Healy, and Astels, 2017a) which represent the level of belongingness of a data point to a cluster, etc. These methods use different ways and assumptions to measure how much a data point differs from its surroundings. For example, LOF assesses a point’s density to its neighboring points, identifying outliers in low-density areas surrounded by higher-density ones. It determines outlier scores for each point by comparing its local density to the average density of its neighbors: A high LOF score denotes an outlier; a low score indicates membership in a dense cluster. Thus, the LOF score can be seen as the reachability distance ratio of a point’s neighbors to its own.

LOCI, instead, calculates an outlier score based on the variation of the local neighborhood, called the multi-granularity deviation factor (MDEF), which can be thought of as the degree of deviation of the density of a point’s neighborhood from the average density of the neighborhoods of the points in the neighborhood. It marks as outliers the samples that have a large MDEF value. LOCI autonomously detects outliers, requiring no user input (i.e. calculates the cut-off to check whether a point is an outlier), and can generate LOCI plots for individual points, providing visual insights into local data density, clusters, and micro-clusters. That is, LOCI not only identifies isolated single outliers but also groups of outlier clusters (aka micro-clusters). Efficient computations of LOCI utilize speed-enhancing approximations.

GLOSH is a method for detecting outliers in data, meaning that they deviate significantly from their surroundings. It can detect both local and global outliers, where local outliers deviate from their local neighborhood, and global outliers deviate from the overall data distribution. This means that GLOSH is capable of identifying outliers that may significantly differ from nearby points without necessarily being outliers on a global scale. GLOSH assigns an outlier score to each point by measuring its density relative to the densities of other points in the same and lower-level clusters. The clusters are obtained using different linkage methods, e.g. by applying a hierarchical clustering algorithm, such as HDBSCAN (McInnes, Healy, and Astels, 2017a), to the data. Points that belong to low-density clusters or that are far from the core of their clusters have high outlier scores. This allows GLOSH to identify global outliers, but also outliers that are only conspicuous in their local context but not globally. GLOSH is similar to previous outlier detection methods such as LOF and LOCI which also use density-based criteria to find outliers.

HDBSCAN itself can also find outliers by computing cluster probabilities of data points based on how long their clusters persist in the condensed tree. Low probabilities indicate outliers or border points. Cluster probabilities and outlier scores use different formulas, as they measure different aspects of clustering: persistence and density, respectively. So, the clustering probabilities center around the degree to which a point *represents* a cluster, whereas the outlier scores center around how effectively a point *aligns* with a cluster. These are different concepts.

Overall, the key difference between LOF, LOCI, and GLOSH is: LOF relies on distance metrics for proximity measurement, LOCI operates on correlation concepts, and GLOSH is centered on hierarchical principles.

Other common methods include One-Class-SVM (Schölkopf et al., 2001) and Isolation Forests (Liu, Ting, and Zhou, 2008) One-Class-SVM is based on the idea of fitting a hyperplane in a high-dimensional feature space that separates the normal data from the origin. It maps the data into a nonlinear space using a kernel function

and a parameter to regulate the fraction of outliers. Isolation Forests are built on the idea of separating anomalies using random feature space partitions. It constructs a decision (binary) tree ensemble that iteratively splits the data along a random feature and a random value. Because anomalies are easier to isolate, they are expected to have shorter average path lengths in the trees than the usual data.

SUOD ( Accelerating Large-Scale Unsupervised Heterogeneous Outlier Detection) (Zhao et al., 2021) is a system that speeds up the detection of outliers in large-scale datasets. It uses three techniques that work well together: random projection to lower the dimensionality of high-dimensional data (data level), pseudo-supervised approximation to simplify complex models (model level), and balanced parallel scheduling to balance the workload in distributed settings (task level). These techniques can be turned on or off as needed. SUOD claims to detect outliers in large datasets fast and accurately.

Numerous further algorithms and techniques exist; a comprehensive benchmark overview is provided by (Han et al., 2022).

In this work, we focus on post-hoc OOD detection methods which do not influence the original training of the model and are applied directly on the outputs of models. Particularly, we demonstrate that OOD instances can be identified by examining the disparities in model calibrations and predictions.

## 6.5 Supervised Clustering Evaluation

**Notice:** Our work and findings on  $\alpha$ Max-B-CUBED (presented in this thesis) have been publicly released on OpenReview-Venue (Guimerà Cuevas and Schmid, n.d.).

Weak Supervision (WS) is a branch of ML in which the model is trained using noisy, incomplete, or inexact annotations instead of complete and accurate labels (Zhou, 2018). In WS, a model is designed to deal with noise and uncertainty in annotations and make the best possible predictions based on the information given. Because producing high-quality labels is typically very costly, weak supervision is often used to generate additional cheaper, but lower-quality labeled data. Inaccurate training data contains defects or erroneous labels. The term "inexact" refers to training data with imprecise labels, such as coarse categories or probabilistic labels. Inexact supervision can result in a model with lesser prediction confidence, but also in misleading conclusions during supervised evaluation of clustering results. Cluster quality evaluation with weak labels, thus, refers to the process of evaluating the performance of a clustering method when the ground truth labels are not fully known, missing, noisy, uncertain, or too generic. "Incomplete" training data refers to data that lacks key information, labels, or characteristics.

There exist different evaluation metrics that address cluster quality evaluation with *weak* labels or uncertainty. E.g., data clustering with partial supervision, where data is neither completely nor accurately labeled, can be approached using a fuzzy clustering-based technique to leverage available data knowledge to supervise the clustering process (Bouchachia and Pedrycz, 2006). The Adjusted Rand Index (Rand, 1971) measures the similarity between the real and predicted (cluster) labels while adjusting for chance; related to accuracy. "Chance" refers to the possibility of achieving a specific outcome by random chance in the context. This involves considering the possibility that the agreement between the true and predicted labels might occur by coincidence, even if the clustering method is not appropriately grouping the data points. Normalized Mutual Information (Press et al., 2007) evaluates the mutual information between the real and predicted cluster labels, normalized by the

entropy of both; where the entropy can be regarded as a measure of uncertainty. The Fowlkes-Mallows Index (FMI) (Fowlkes and Mallows, 1983) computes the geometric mean of precision and recall between true and generated clusters. Uncertainty in the true labels might affect the accuracy of the clustering results because it may be difficult to accurately assign data points to their true labels if the actual labels are not clearly defined or known. FMI compensates for this uncertainty by calculating precision and recall using both the number of correct and incorrect predictions.

$B^3$  can be seen as a "precision and recall metric" for clusters. Following (Amigó et al., 2009), two items that share a category are correctly related if and only if they occur in the same cluster. An item's  $B^3$  precision is the fraction of objects in its cluster that have the item's category. The average precision of all items in the distribution is used to calculate the overall  $B^3$  precision. The  $B^3$  recall is analogous. The  $B^3$  clustering algorithm has gained a lot of attention and received improvements and refinements to adapt to different situations. The adapted  $B^3$  metrics (Moreno and Dias, 2015) were proposed for imbalanced datasets. The authors claim that the original family of  $B^3$  metrics is not well adapted when datasets are imbalanced. The *Cluster-Identity-Checking Extended  $B^3$*  (CICE- $B^3$ ) (Rosales-Méndez and Ramírez-Cruz, 2013) was proposed as a new evaluation measure for overlapping clustering algorithms consisting of a new approach to determining precision, recall, and the F-measure, which analyzes object pairings that co-occur in clusters and/or classes. Critics have also pointed out that  $B^3$  overestimates performance by crediting clustering for placing an element within its own cluster (Heusden, Kamps, and Marx, 2022); which prompted the revised metric "Elements Like Me (ELM)".

In this work, however, we further show how the standard  $B^3$  metric may not perform optimally or lead to misleading comparisons when assessing different clustering outcomes under label uncertainty; e.g. when dealing with coarse labels. To overcome this issue, a modified mathematical metric formula for  $B^3$  is suggested that integrates cluster group super-aggregation into its scoring function.

## 6.6 Manifold Clustering

As already highlighted, cluster analysis is a crucial and key field in ML; and explainable AI is a very active area of research and experimentation (Gunning et al., 2019; Saisubramanian, Galhotra, and Zilberstein, 2020; Xu et al., 2019). Cluster interpretability and explainability refer to the ability to understand and interpret the results of a clustering algorithm. A major challenge here is *how* to effectively, transparently, and reliably explain the results and automate this process, especially since unsupervised numeric clustering typically cannot be directly understood by humans without any further analysis or interpretative tools. Of course, manually analyzing each cluster is not scalable and tedious, and randomly selecting a subset of instances from a cluster and extrapolating their characteristics to the entire cluster is obviously not effective at all. This is related to a concept referred to as Visual Interpretation (VI): the transformation of information into interactive and visual forms, such as graphs or images, to facilitate understanding and (manual) analysis (Healy, 2018). However, VI has many limitations, including being non-formal (in a mathematical sense), unreliable, subjective, and time-intensive. Consequently, a systematic and automated approach is often imperative for accurately explaining the results yielded by AI- or clustering algorithms. Automated feature-based model interpretability tools, such as SHAP (Lundberg and Lee, 2017) and LIME (Ribeiro, Singh, and Guestrin, 2016), can help understand the characteristics that drive a given

cluster assignment, making the results more interpretable, e.g. by highlighting pixels or image sections that are most important to the cluster assignment; the explanation can be a simple heat map over the pixel score values.

Manifolds are a crucial concept in NNs, and the knowledge encoded in latent semantic manifold spaces has been extensively researched and exploited for clustering: McConville et al., 2021 proposed a combined training of clustering and non-clustering loss that first computes the embedding using an autoencoder, then searches for underlying manifolds e.g. using UMAP (McInnes, Healy, and Melville, 2018), and finally performs clustering on the newly learned latent manifold representations. SMCE (Elhamifar and Vidal, 2011) is an algorithm that performs clustering and dimensionality reduction of data in multiple nonlinear manifolds, handling close, non-uniform sampling, and holes, and estimates intrinsic dimensions. Deep Embedding Networks (Huang et al., 2014) employ autoencoders to derive reduced representations from raw data, applying constraints to maintain locality and group sparsity, which are then clustered. Robust Continuous Clustering (Shah and Koltun, 2017) learns condensed data representations by integrating simultaneously representation learning and clustering. Similarly, Shah and Koltun, 2018 also performs joint nonlinear dimensionality reduction and clustering by embedding data into a lower-dimensional space using an autoencoder. Leveraging knowledge about manifolds was also proposed in DMMC (Chen, Lv, and Zhang, 2017), which discovers clusters for unlabeled data residing on distinct manifolds. It is motivated by the assumption from Rodriguez and Laio, 2014 that close points are on the local of a manifold and should have comparable representations, i.e., cluster centers are surrounded by neighboring points with lower local density and are relatively far from any points with greater local density.

More recently, Li et al., 2022 argued that performing manifold clustering with neural networks requires two fundamental ingredients: a domain-specific constraint that ensures *manifold identification*, and a *learning method* for embedding manifolds to linear subspaces in the feature space. This was e.g. used in NMCE (Yu et al., 2020), a method that combines data augmentation with principles of MCR<sup>2</sup> and outperforms autoencoder-based deep subspace clustering. NMCE follows the principles of respecting domain-specific constraints, not collapsing manifold embeddings, and linearizing and separating identified manifold embeddings.

In this thesis, we put attention on exploring manifold aggregations of latent representations to interpret clusters on a descriptive textual approach. This is useful e.g. for generating tags for image clusters, identifying important words for clustering decisions, or calculating cluster similarities based on semantic textual similarities. Our results demonstrate that even basic manifold aggregation methods, especially when coupled with a population-based score-function selection, can create concise and representative cluster descriptions. This is achieved by leveraging the implicit semantic understanding embedded in clusters within the inherent latent manifold structure of large vision and language models.

Moreover, we also explore clustering via manifold identifications by determining approximation errors and similarities in the Taylor Approximation of the manifold space; proposing a new approach and algorithm for clustering manifolds.

## 6.7 Evaluating Generated Content against References

Comparative text analysis plays a vital role in evaluating text generation models, benchmarking their performance, ensuring high quality, estimating human parity,

and optimizing the models. Evaluating the quality and similarity of text data is non-trivial; yet, as language models become more effective, novel robust metrics have emerged.

In this work, we will emphasize and center around three deep learning methods to assess the quality (i.e., similarity) of a model’s generated prediction text against a (set of) reference text(s); namely *BERT-score* (Zhang\* et al., 2020), *BART-score* (Yuan, Neubig, and Liu, 2021), and *BLEURT-score* (Sellam, Das, and Parikh, 2020). BERT-score uses BERT’s pre-trained contextual embeddings and cosine similarity for evaluation; it provides precision, recall, and F1 measures. BART-score uses the BART pre-trained model for text generation-based evaluation; it offers different perspectives such as coherence, semantic coverage, informativeness, fluency, or actuality. BLEURT-score measures how well a candidate fluently conveys the intended meaning of a reference. It operates as a trained regression model metric.

### 6.7.1 BERT-score

BERT-score (Zhang\* et al., 2020) leverages BERT’s contextual embeddings, aligning candidate and reference sentences via cosine similarity. It was shown to correlate with human judgment. It calculates precision, recall, and F1 for text quality assessment. It is designed to measure the similarity between a reference (ground truth) sentence and a candidate (generated) sentence. BERT-score is a supervised matching metric that measures the semantic equivalence between a reference and a target. It employs token-level matching functions within the distributed representation space, leveraging the continuous vector spaces of the word embeddings; i.e., it calculates token similarity by utilizing contextual embeddings derived from BERT.

### 6.7.2 BART-score

BART-score (Yuan, Neubig, and Liu, 2021) is a metric designed for evaluating text generation. It is straightforward in concept and has shown good empirical performance, surpassing other top-scoring metrics. BART-score leverages BART, an encoder-decoder pre-trained model, to convert the generated text to or from a reference output or source text. The underlying assumption is that models trained to perform this conversion will yield higher scores for better-generated text. That is, it is expected that: the better (e.g. fluent, semantic similar, etc.) a given text is to a reference text, the better the scores will be; and so the evaluation is seen as a *text generation* problem. BART-score is claimed to be resource-efficient in terms of parameters and data; its architecture does not include any additional parameters beyond what is used during pre-training. BART-score is trained in an unsupervised manner; i.e. it does not require any human judgments for training/learning. Overall, BART-score supports various evaluation scenarios based on different generation directions (see Yuan, Neubig, and Liu, 2021 for more details):

- **BART-Faithfulness:** Measures the likelihood of generating the hypothesis from a source document. It can evaluate factuality, relevance, coherence, and fluency of the target text.
- **BART-Precision:** Assesses how likely a hypothesis could be constructed based on the gold reference. It is useful for precision-focused evaluation.
- **BART Recall:** Quantifies how easily a gold reference could be generated by a hypothesis.

- **BART-F-score:** Considers both Precision and Recall. It provides a broad evaluation of the semantic overlap between reference texts and generated texts.

So, therefore, BART-score is a measure of how well a candidate sentence can be generated from a source sentence using the BART model. It's important to understand that the score is not intended to return a probability of one when the candidate and reference sentences are identical. Instead, the score depends on the quality of the BART model's training and the similarity between the candidate and reference sentences in terms of content and style.

### 6.7.3 BLEURT-score

BLEURT (Sellam, Das, and Parikh, 2020) is an ML-based metric; also typically used to evaluate Natural Language Generation systems. It takes a sentence pair as input: a reference (typically human-generated) and a candidate (machine-generated). The score ranges between 0 and 1 (approximately); and indicates the extent to which a candidate is fluent and conveys the meaning of the reference. BLEURT captures non-trivial semantic similarities between sentences, providing a robust evaluation. As stated in the original paper, BLEURT was introduced as a metric for generating text based on references in *English*; utilizing end-to-end training (future research is expected to focus on expanding its evaluation capabilities to include multilingual evaluation). Overall, BLEURT is a supervised metric in the sense that it requires human judgments to train; trained as a regression model.

## 6.8 Text Generation Strategies

*Decoding* is the process of selecting output tokens to generate text. There are several text decoding strategies that can be used to generate text from a language model. These strategies are used to select the most likely sequence of words from a set of possible outputs. Some common text decoding strategies (Wolf et al., 2020b) include:

- **Greedy-search:** Selects the word with the highest probability at each time step.
- **Contrastive-search:** Optimizes for a diversity-augmented objective.
- **Multinomial sampling:** Randomly selects the next token based on the model's probability distribution.
- **Beam-search:** Keeps track of multiple potential sequences at each step.
- **Beam-search multinomial sampling:** Combines beam search with multinomial sampling.
- **Diverse beam-search:** Generates a diverse set of beam sequences by optimizing for a diversity-augmented objective.

For sampling strategies, one can therefore stochastically generate "multiple" candidate sequences (multi-sequence sampling).

## 6.9 Large Language Models: Prompt Engineering

To recap: Large Language Models (LLMs) are sophisticated DL algorithms that excel in natural language processing tasks (Kaddour et al., 2023). Their large model size is made possible by AI accelerators that can process vast amounts of text data efficiently (Reuther et al., 2020). LLMs are essentially artificial DNNs that can have billions to trillions of weights and are trained through self-supervised and semi-supervised learning. The widely used Transformer architecture (Vaswani et al., 2017) is integral to LLMs.

LLMs can generate human-like responses to **prompts** given by a user. LLMs are (pre-) trained on huge datasets of text, often containing billions of words, using state-of-the-art DL techniques. LLMs revolutionized natural language processing and have many applications and use cases, including classifying, translating, predicting, and generating text, summarization, question answering, and even writing software code. They are active across many different sectors, including healthcare, finance, and entertainment; playing roles such as chatbots, and AI assistants. Famous examples of LLMs are the GPT models from OpenAI (OpenAI, 2023), PaLM from Google (Chowdhery et al., 2022), LLaMa from Meta (Touvron et al., 2023), and Falcon from UAE's Technology Innovation Institute (Penedo et al., 2023).

A *prompt* is a "message" that guides and influences the AI model in carrying out its task; typically in the context of text- or image models. The main purpose is to generate text that a generative AI model can understand. "In-context learning" enables the model to gain knowledge from brief prompts. Prompt engineering (for a full and extensive survey and overview, see Saravia, 2022; Weng, 2023) is a key concept when working with LLMs. It consists of (carefully) selecting and designing the appropriate words, phrases, symbols, and formats to guide the model.

Prompts act as inputs or queries users give to obtain specific responses from a model. It affects the response length and detail. Efficient prompt engineering is crucial in achieving the desired outcomes with LLMs. It can require creativity and careful attention to detail, involving delimiters, output formats, condition checks, few-shot prompting, and task completion steps. Prompt engineering is an important task/skill for anyone working with LLMs; which is experiencing more relevance as more organizations implement LLMs to automate tasks and improve productivity. A well-defined prompt can assist in getting the most out of their LLMs and is, thus, important. Beside prompts, temperature, top-k, top-p, frequency penalty, and presence penalty are examples of other controls that impact model output behavior.

For example, if you ask the model, "What are the benefits of plants?" it might ask for clarification because there are many types of plants with different benefits. However, if you specify the type of plant, such as "What are the benefits of aloe vera?", the model would generate a specific response about the benefits of aloe vera. Therefore, the way a prompt is formulated directly influences and determines the model's response. And so, it is important to be clear and specific in the prompt to guide the model to generate a desired response type. This is known as prompt engineering, which is used to direct LLMs to produce specific outcomes *without* altering their weights. Prompt engineering is an empirical discipline, implying that its methods can produce varied results on different models; typically requiring extensive testing and heuristic approaches (Weng, 2023).

Multiple approaches exist for prompt design/engineering. E.g., two common approaches for language model fine-tuning are: zero-shot and few-shot learning. The former involves inputting the task text directly into the model but may result

in imprecise or ambiguous responses. In contrast, few-shot learning includes high-quality examples to improve model outputs. However, there are limitations to the length of input and output text, and the choice of prompt format, training examples, and order can greatly impact performance; e.g., an extensive prompt increases the actual text, thus, ultimately decreasing the amount of free-text that can be inputted or analyzed. In general, to avoid bias in few-shot prompt learning, it is suggested to select diverse and relevant examples in random order and to choose an order that prevents extremely unbalanced or overconfident predictions when the validation set is limited. Also, increasing model size or adding more training examples may not necessarily reduce variance in different permutations of examples, and the effectiveness of specific example orders may vary depending on the model (Lu et al., 2021).

Few-shot examples help a model understand task instructions but require more tokens and, again, have input length limitations due to context constraints. This leads to the question of why not simply provide direct task instructions through instruction prompting. However, accurately describing a task may be difficult; and may require specific and precise information. Overall, one should then emphasize actions to be taken (positive actions) rather than actions to avoid (negative actions) (Saravia, 2022). Few-shot learning was also combined with instruction prompting via *in-context instruction learning* (Ye et al., 2023).

Self-consistency sampling (Wang et al., 2022b) is a technique that involves generating multiple candidate outputs and then selecting the best one based on specific criteria, heuristics, quality measures, etc.; which one(s) to use depend on the actual task. That is, to evaluate the quality of the outputs, a metric or measurement system is needed. E.g., one approach for selecting the best output is to use ensemble methods, e.g. combining multiple metrics or criteria.

Chain-of-thought (CoT) prompting (Wei et al., 2022) breaks down reasoning into a series of concise sentences, providing step-by-step logic and reasoning. The result is an output of reasoning chains or rationales that guide the user towards a final answer. CoT is particularly advantageous for complex reasoning tasks, especially when working with extremely large LLMs. However, for simple tasks, the benefits of CoT prompting are minimal. Interestingly, it was observed (Fu et al., 2022) that when separating CoT steps, the newline '\n' symbol is more effective than using 'step i', a dot '.', or a semicolon ';'. Additionally, using 'Question:' instead of the abbreviation 'Q:' was also beneficial.

*Automatic prompt design* is an interesting concept for automatically optimizing the series of prefix tokens to increase the likelihood of obtaining an intended output for a given input. By treating prompts as trainable parameters, they can be optimized through known techniques; such as via gradient descent on the embedding space. Several methods, such as AutoPrompt (Shin et al., 2020), Prefix-Tuning (Li and Liang, 2021), P-tuning (Liu et al., 2023), Prompt-Tuning (Lester, Al-Rfou, and Constant, 2021), and APE (Zhou et al., 2022b), have been developed for automatic prompt design. In fact, even an LLM itself can be used and utilized to generate or propose suitable prompts for a given example (or set of examples).

A prompt may consist of multiple different parts (Saravia, 2022):

1. **Instruction:** The specific task for the model to perform.
2. **Context:** External information to improve responses.
3. **Input Data:** Input or question to find a response for.
4. **Output Indicator:** Type/format of output.

A full prompt example may therefore be the following example:

- **Instructions:**
  - Generate a summary of the latest developments in AI research.
- **Context:**
  - The audience for the summary consists of non-experts in AI.
- **Input Data:**
  - The latest developments in AI research.
- **Output Indicator:**
  - Present the summary in bullet points.
  - The summary must not be longer than 350 words.
  - Use language that non-experts in computer science can easily understand.

So overall, as a general guideline when creating prompts for models, it is considered useful to start with simple prompts and only then gradually add more elements until one achieves more optimal results. It is important to experiment and iterate with prompts for better outcomes. Specificity, simplicity, and conciseness can enhance the effectiveness of prompts, as well as breaking down large tasks into simpler subtasks. Short clear commands such as "Write", "Classify", "Summarize", or "Translate" can already strongly guide the model's actions. It is sensible to experiment with different keywords and data and provide specific and relevant context for the task at hand. Providing examples is, as explained, an intuitive yet effective approach to achieving desired outputs. It is key to avoid imprecise language; instead being direct and specific is important. This includes avoiding negative phrasing and focusing on what the model should do rather than what it should *not* do.

## 6.10 Representative Sampling in Data Streams

### 6.10.1 Background on Reservoir-Sampling

Many different variants and implementations of Reservoir Sampling (RS) exist, each with its own optimizations, improvements, and intended use cases. Our objective here is to present a brief and concise summary of the different common algorithmic sampling methods; while we again encourage the reader to review the respective literature (Bressan and Lu, 2009) for a more comprehensive and detailed understanding. Overall: RS enables progressive sampling in a single pass in both, dynamic and large static datasets.

#### Reservoir-Sampling

Reservoir Sampling (Vitter, 1985) is a probabilistic algorithm designed for randomly selecting  $k$  samples from an indefinitely long data stream of unknown length. In this context, the parameter  $k$  may signify the memory constraint, specifying the total number of items to retain. RS maintains a reservoir, which is a static collection such as an array or list with a fixed size  $k$ . At any given moment, the reservoir always

contains a random sample of  $\leq k$  elements drawn from the input stream in a uniform and random manner.

Mathematically, RS can be formally described as: Let there be a data stream denoted as  $S$  with a length of  $N$  (which may be finite or infinite). The objective is to stochastically select a sample of size  $k$  (where  $k \leq N$ ) from the stream to be stored in an array  $R$  of length  $k$ . RS begins by initially populating  $R$  with the first  $k$  elements from  $S$ . Then, for all  $k < i \leq n$ , the following steps are performed:

1. Generate a random integer  $j$  within the range 1 to  $i$ .
2. If  $j \leq k$ , substitute the  $j$ -th element of  $R$  with the element  $S[i]$  from the stream.

At any given time-step,  $R$  contains a random sample of  $\leq k$  elements, chosen *uniformly* and independently at random from the  $S$ ; where each element had an equal probability of  $1/i$  of being included in  $R$ .

### Biased Reservoir-Sampling

In biased reservoir sampling, a random sample of size  $k$  from a population of size  $N$  is selected, where each element in the population is assigned a weight or probability  $p_i$ . The selection is biased to reflect these probabilities, ensuring elements with higher weights are more likely to be included. The RS algorithm was extended (Chao, 1982) to handle arbitrary sampling probabilities.

### Decaying Reservoir Sampling

Decaying reservoir sampling (Aggarwal, 2006; Cormode et al., 2009) is also meant to obtain a random sample of fixed size from a continuously arriving data stream. However, it works by assigning *decaying* probabilities of keeping the elements in the reservoir, with a higher probability given to more recent elements. This ensures that recent elements are more likely to be included and remain in the reservoir sample than older elements, while still allowing for older elements to have a (small) chance of being selected; controlled by an exponential decaying factor, which determines the rate at which the probabilities decrease over time.

### Sliding Window Sampling

Sliding Window Sampling continuously samples a data stream using a fixed-size window (Braverman, Ostrovsky, and Zaniolo, 2009; Jayaram, Woodruff, and Zhou, 2022; El Sibai et al., 2015). It differs from normal reservoir sampling by maintaining a sample of the most recent data points given a sliding window size. RS may here be limited within that sliding window.

### Quantiles in Data Streams

A quantile query (Buragohain and Suri, 2009), formally defined, is a statistical analysis that seeks to identify a value, denoted as  $v$ , within a dataset  $S$ , such that its rank, denoted as  $\text{rank}(v, S)$ , aligns with a specified quantile  $\phi \in [0, 1]$ . Mathematically, this can be represented as  $v = \arg \min_{v' \in S} |\text{rank}(v', S) - \phi \cdot |S||$ , where:  $v$  is the desired value,  $\text{rank}(x, S)$  is the rank of a data point  $x$  in the sorted dataset  $S$ ,  $\phi$  is the target quantile, with  $\phi \in [0, 1]$ , and  $|S|$  represents the size of the dataset  $S$ . In essence, a quantile query aims to find a value within the dataset that corresponds to a specific quantile of the data distribution.

## Spatio-Temporal Trajectories

A spatial-temporal trajectory  $T$  in an  $n$ -dimensional space vector is a mathematical construct representing the movement of an object or entity through space and time. It can be defined as a sequence of data points  $(\vec{z}_i, t_i)$ , where  $\vec{z}_i$  is an  $n$ -dimensional vector representing the object's spatial location at time  $t_i$ .

Mathematically, a spatial-temporal trajectory in an  $n$ -dimensional space vector can be represented as:  $T := ((\vec{z}_1, t_1), (\vec{z}_2, t_2), \dots, (\vec{z}_N, t_N))$ , where  $N$  is the (current) number of data points in the trajectory; and  $t_i$  is strictly-monotonically increasing.  $T$  may extend infinitely, constituting an unbounded series of spatial locations along with respective timestamps  $(\vec{z}_i, t_i)$ ; over a continuous stream of points over time.

### 6.10.2 Related Work on Data Stream Sampling

Again, processing and sampling of data streams is an important topic and multiple extensive reviews and surveys exist in the literature for data streams (Almeida et al., 2023; Gaber, Zaslavsky, and Krishnaswamy, 2007; Haas, 2016; Kolajo, Daramola, and Adebiyi, 2019; Sibai et al., 2016). Data-stream sampling combines traditional database sampling techniques with adaptations for streams of infinite lengths. Commonly, two approaches are stationary and (sliding) windows. Following Haas, 2016, a stationary window has fixed endpoints, which could either be specific timestamps or positions within the stream. In contrast, a sliding window has dynamic endpoints that move alongside time. Here, sampling becomes more intricate since expired elements must be removed, making it a more challenging task to sustain an optimal sample size (Haas, 2016).

The authors also mention so-called "generalized" windows; here a data stream consists of operations that can insert and delete items into/from the window. In fact, a standard sliding window is a special case where items are deleted in the same order that they are inserted.

Sampling from a *stationary* window can be achieved e.g. through techniques such as Bernoulli Sampling or Poisson Sampling (Särndal, Swensson, and Wretman, 2003). However, a notable drawback of both Bernoulli and Poisson sampling methods is the inherent uncontrollable (non-deterministic) variability in the sample size. Another naive approach is using Stratified Sampling (Parsons, 2014). It involves partitioning the window into distinct intervals, referred to as "strata", and extracting a predetermined sample size from each of these strata. E.g., the simplest method for a sample of size  $k$  is to pick every  $m$ th element (where  $m$  equals  $N/k$ ); this approach lacks statistical context for the entire window and can lead to unrepresentative samples if the incoming data has periodicity matching the sampling rate (Haas, 2016). Biased Sampling by "halving" (Haas, 2016) is a statistical technique for obtaining a biased sample from a larger group. It works as follows: The population is split into  $L$  strata, each with  $m = N/L$  elements. A running sample  $R$  of size  $k$  is started with a simple random sample (SRS) from the first stratum. In each step, half of the elements in  $R$  are randomly swapped with elements from an SRS of the next stratum. The likelihood of an element from a stratum being in the final sample decreases geometrically as its index decreases.

Sampling from a *sliding* window expires older elements falling outside the window range and focuses on the most recent data. To maintain a sample of a specified size, elements that expire must be immediately removed. It can further be distinguished between sequence-based windows, which e.g. hold the  $N$  most recent elements, and timestamp-based windows, which store elements received within the

past  $t$  time units (Babcock, Datar, and Motwani, 2002). However, it is important to note that in generalized windows, elements can be removed in any order. Additionally, the sliding window size for "valid" elements to be sampled can be larger than the actual size of the reservoir. Still, in the case of standard sliding windows (from "left to right"), which eliminate the oldest elements and include the newest elements, the window size corresponds obviously to the sample size.

Since memory is limited and smaller than the total incoming (infinite) data, precise quantile computation is not possible, and the best achievable solution is an approximation since any data not explicitly stored is irretrievably lost (Buragohain and Suri, 2009). Recent research has focused on achieving provable-quality approximations of quantiles, aiming to provide epsilon-approximate quantile summaries of a sequence of data elements. These summaries allow quantile queries within a certain precision, primarily defined by the size of the summary data structure. Random sampling, however, though easy to implement, requires a relatively large sample size to achieve an expected level of approximation accuracy. There are several variants of quantile summaries, including those for distributed data streams, sliding windows, biased estimations, and duplicate-insensitive quantiles (Buragohain and Suri, 2009), and many approaches for maintaining quantile summaries in data streams (Chen and Zhang, 2020; Liang, Li, and Liu, 2019; Lin et al., 2004).

On the other hand, trajectory sampling is a method that involves selecting a subset of data points from a (very) long or infinite trajectory to extract a meaningful representation while preserving essential characteristics. This is useful for (1) reducing computational complexity, as analyzing the complete trajectory can be computationally intensive, (2) data reduction, as the full trajectory may contain redundant or irrelevant information, and (3) facilitating visualization and interpretation by working with a much smaller subset of data. Sampling from infinite trajectories is directly related to data streams, especially in the context of real-time analysis; involving continuous, ongoing processing as it arrives. Representative trajectory (i.e. data stream) sampling is also an important concept for measuring the similarity or distance of different (infinite) trajectories (Su et al., 2020; Tao et al., 2021; Toohey and Duckham, 2015; Magdy et al., 2015; Wang et al., 2013), and particularly of interest in the context of DL (Li et al., 2018b; Fang et al., 2022; Yang et al., 2021).



## Chapter 7

# Methods

In this chapter, we present the novel methods and strategies developed to effectively address and solve the core problem discussed in this thesis work. Building upon the concepts outlined in the previous Chapters 2 - 6, this section assumes a foundational understanding of ML methodologies and terminologies. A concise summary of all contributions was provided in Chapter 1 [Contributions].

### 7.1 Neural Networks Based on Complex Numbers with Weights Constrained along the Unit Circle

**Notice:** Our work and findings on complex-valued NNs (presented in this thesis) have been published in Guimerà Cuevas, Phan, and Schmid, 2023. It was inspired and builds upon our previous research (Guimerà Cuevas and Phan, 2021), but differs significantly as it has been substantially improved and modified (both the complex architecture and its mathematical framework); thus, constitutes novel contributions. In particular, this novel contribution has been accepted for publication, after peer review. The Version of Record is available online at: [https://link.springer.com/chapter/10.1007/978-3-031-33374-3\\_28](https://link.springer.com/chapter/10.1007/978-3-031-33374-3_28). Use of this Accepted Version is subject to the publisher's Manuscript terms of use. Therefore, please refer to Guimerà Cuevas, Phan, and Schmid, 2023 for specific details on sections:

#### 7.1.1 Convolutional Layer

#### 7.1.2 Numerical Stability

#### 7.1.3 Weight Initialization

#### 7.1.4 Alternative Normalization Correction Factor

#### 7.1.5 Enhanced Complex Neurons with Linear Weight Scaling

##### Take-Away

To mitigate problems associated with weight regularization, weight suppression, and information loss due to zero weights, it is possible to utilize **complex-valued NNs instead of their real-valued NNs**. This alternative not only addresses the mentioned issues but also enhances the network's expressiveness, albeit at the cost of increased computational complexity.

### Keywords

Complex-valued Neural Networks • Weight Regularization • Expressiveness

## 7.2 Data Representation & Encoding

We have already introduced the raw structure of the input data in Chapter 4.1. In this section, we will outline our encoding method (i.e. input transformation/preparation) for hierarchically encoding HDBs into special tree structures based on graphs, which will then serve as the input for our model.

### 7.2.1 Hierarchical Data Interpretation

Given multiple HDB entities with various relationships, both to groups and recursively to subgroups, we can interpret and represent this inherent structure of such hierarchical arrangements as "trees" (Figure 7.1). A given tree structure can here take any arbitrary shape, specifically with a varying number of (sub-) nodes at each level. It may also be entirely asymmetrical, display inconsistent depths along sub-paths, and encode multiple diverse (heterogeneous) features for different node types. That way, we can easily model  $1 : n$  relations in a hierarchical manner where the order of sub-nodes (aka child nodes) is irrelevant, i.e. undefined and holds no significance; which is exactly what we want since we are dealing with quantitative relations (this is very important).

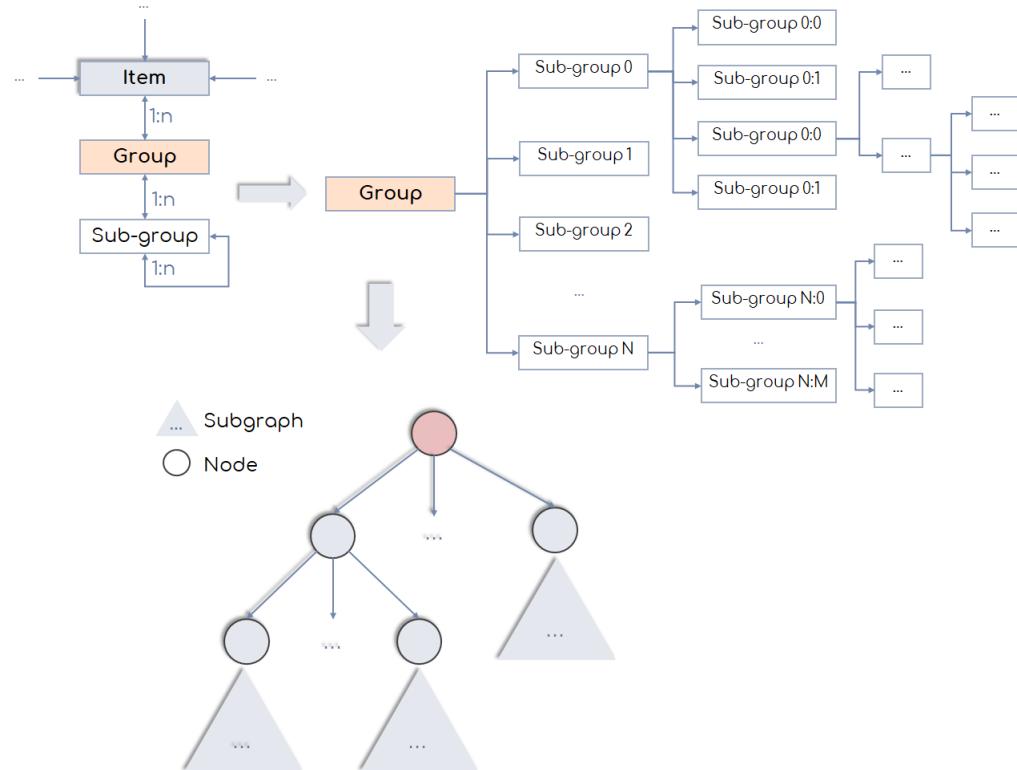


FIGURE 7.1: Interpreting raw data as graph structures.

Each tree node may then hold (i.e. store) any number and type of features (Figure 7.1), such as categorical values, numerical values, etc. Therefore, different nodes

hold different features. E.g., one node may represent a specific sensor; and different sensors collect different features. And so, not only the tree structure is important, but also the feature values. In other words, the meaning/representation of the tree structure extends beyond its topology and also encompasses the specific feature values associated with each node.

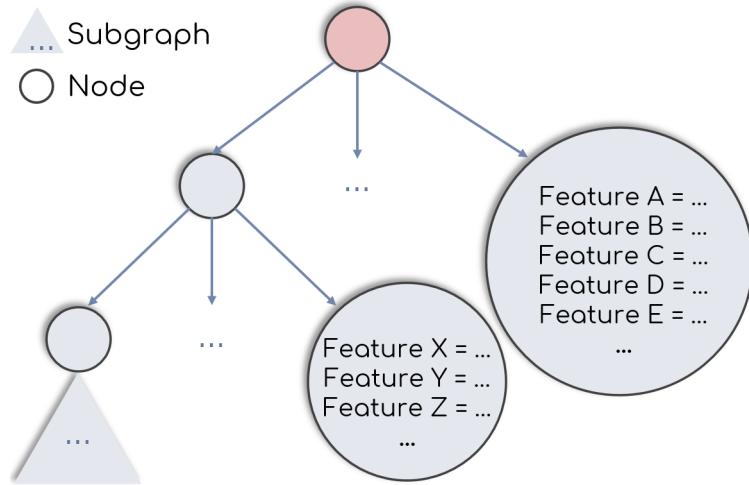


FIGURE 7.2: Visualizing feature storage within tree-nodes.

However, the encoding in Figure 7.1, as presented, would have a downside: not all child nodes must have the same features (and feature dimensions); and so feature aggregations cannot be directly performed. Ideally, nodes under a common parent node would share the same features, and nodes under different parent nodes can/- would have distinct feature types, counts, and values (Figure 7.2).

Therefore, we further propose grouping nodes from the same HDB table entity under a common *virtual* node; guaranteeing so that all sub-nodes of any given parent node have equivalent features. An illustration of transforming an HDB into a tree-like graph with virtual nodes is given in Figure 7.3.

The important key points of the encoding can, hence, be summarized as:

- **Hierarchical Structure Representation:**
  - HDB entities with multiple diverse relationships can be interpreted as hierarchical structures.
  - Representation of this structure is akin to "trees" with arbitrary shapes, varying node counts, asymmetry, and inconsistent depths.
- **Node Features:**
  - Each tree node can store various features and feature types, e.g. categorical and numerical values.
  - Different nodes may represent different entities; such as sensors collecting different features.
- **Importance of Tree Structure and Features:**
  - The meaning of the tree structure goes beyond its topology; it is also influenced by the feature values assigned to each node.

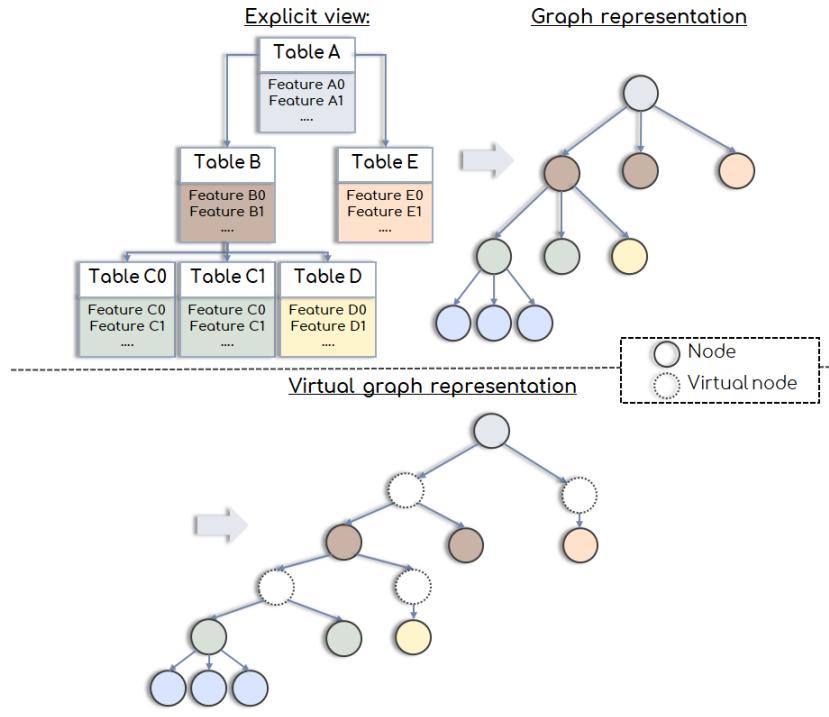


FIGURE 7.3: Top-left: Original HDB (Hierarchical Database). Top-right: Corresponding tree-like graph representation. Bottom: Final transformation with virtual nodes ensuring equal color distribution among child nodes. Different colors denote distinct DB tables.

- Both the structure and feature values shape the overall representation.

- **Virtual Nodes:**

- Nodes from the same HDB table entity are grouped under a common "virtual" node.
- This ensures that all sub-nodes of a parent node have equivalent features; and so, feature consistency.

### 7.2.2 Tree-Data Encoding

Now that we have a high-level picture of the encoding scheme, the next step is the *actual* encoding of the tree structure, in particular, the encoding of the individual nodes. The  $1 : n$  relations of the HDB are represented hierarchically as trees. Each node in a respective HDB tree structure is assumed<sup>1</sup> to have all the features of its corresponding "DB table", although not all tables must exist as nodes, and each table can have an arbitrary number of sub-tables (i.e. child nodes) and features; including different feature types and dimensions. DB tables on the same hierarchical level are separated and rejoined by virtual parent nodes, such that every child node of a particular virtual node now belongs to the same DB table. Each node further contains a dynamic feature vector representing all of its sub-trees. This encoding pattern is designed recursively. A node's representation consists of two components:

<sup>1</sup>In Section 7.2.3, we propose an approach to encode non-hierarchical entities. This can be used for and applied to cases where nodes only have a subset of a DB table's features. Individual features are transformed into nodes, addressing missing features as they become missing nodes.

- A **feature representation**  $\vec{v}_{feat}$ ,
- and a **sub-embedding representation**  $\vec{v}_{sub}$ .

The feature representation vector stores the encoding of features from the corresponding DB table; as well as a *node identification feature* to assure the distinction of different node types or node instances (i.e. node graph paths, or node instances) as part of the feature vector. A node's complete encoding is therefore a representation of itself plus that of its sub-graph. Formally, let  $\omega \in \Omega$  denote a node instance of all possible nodes  $\Omega$  with respective sets of direct child nodes  $\theta_i \in \Omega_\omega$ . Note, that any node may be the root node of a sub-graph. We will denote the whole set of nodes within its corresponding sub-tree (including the root  $\omega$ ) as  $\Omega_\omega^\triangle$ .

Let  $\vec{v}_\omega$  be the feature and sub-embedding combination, defined as:

$$\vec{v}_\omega := \vec{v}_{\omega,feat} \oplus \vec{v}_{\omega,sub} \quad (7.1)$$

where  $\oplus$  is the concatenation operation. Nodes without any features (e.g. the root node) have consequently  $\vec{v}_\omega \triangleq \vec{v}_{\omega,sub}$ . The challenge and art here is determining a representative  $\vec{v}_{\omega,sub}$  for all sub-nodes; Features  $\vec{v}_{\omega,feat}$  are fixed per definition. Let  $U_\omega := \{\vec{v}_{\theta_1}, \dots, \vec{v}_{\theta_{|\Omega_\omega|}}\}$  be the union of all sub-embeddings for all child nodes  $\theta_i \in \Omega_\omega$  of a given node  $\omega$ :

$$U_\omega \triangleq \left\{ \begin{array}{c} \vec{v}_{\theta_1} \\ \vec{v}_{\theta_2} \\ \dots \\ \vec{v}_{\theta_{|\Omega_\omega|}} \end{array} \right\} = \left\{ \begin{array}{c} [\vec{v}_{\theta_1,feat} \oplus \vec{v}_{\theta_1,sub}] \\ [\vec{v}_{\theta_2,feat} \oplus \vec{v}_{\theta_2,sub}] \\ \dots \\ [\vec{v}_{\theta_{|\Omega_\omega|},feat} \oplus \vec{v}_{\theta_{|\Omega_\omega|},sub}] \end{array} \right\} \quad (7.2)$$

Note that the *child-collection*  $U_\omega$  has no order. It is very important to provide a permutation-independent mapping since a graph has no "ordering" of its child nodes, just as a DB has no "ordering" of its sub-tables. Otherwise, we would have many different representations for the same entity, which is obviously unacceptable. An illustration of a node's composition is given in Figure 7.4 below:

A *child-matrix* representation of  $U_\omega$  be denoted as  $\mathbf{U}_\omega \in \mathcal{R}^{|\Omega_\omega| \times |\vec{v}_\theta|}$ . The number of columns in the child-matrix  $\mathbf{U}_\omega$  corresponds to the encoding size  $|\vec{v}_\theta|$  of the concatenation of the feature- and child-encoding (which is equal for all nodes within a common parent; e.g. virtual node). The number of rows corresponds to the number of child nodes and can vary between nodes. Leaf nodes have no child nodes and so their node encoding is their feature encoding. The actual row permutation of  $\mathbf{U}_\omega$  is irrelevant; the final embedding encoding will (and must) be permutation independent, again, just as a DB has no concept of "ordering" for its (sub-)tables; there should only be one deterministic representation per instance<sup>2</sup>.

Let  $f_\Omega$  denote a function for any instance  $\omega \in \Omega$  that returns the embedding representation of a given child-collection, so that any child-matrix with arbitrary ordering of rows has the same embedding as the child-collection, i.e. a mapping, so that  $\forall_{\mathbf{U}_\omega, \mathbf{U}'_\omega} : f_\Omega(\mathbf{U}'_\omega) = f_\Omega(\mathbf{U}_\omega) =: f_\Omega(U_\omega)$  holds for any two explicit matrix representations of  $U_\omega$ . Using the recursive definition  $\vec{v}_{\omega,sub} := f_\Omega(\mathbf{U}_\omega)$ , it follows:

$$\vec{v}_{\omega,sub} \triangleq f_\Omega(\bigcup_{\theta \in \Omega_\omega} \{\vec{v}_\theta\}) = f_\Omega(\bigcup_{\theta \in \Omega_\omega} \{[\vec{v}_{\theta,feat} \oplus \vec{v}_{\theta,sub}]\}) \quad (7.3)$$

<sup>2</sup>Ensuring consistent and deterministic embedding representations is essential for stability, effective training, and generalization, similarity search, clustering, instance discrimination, etc.

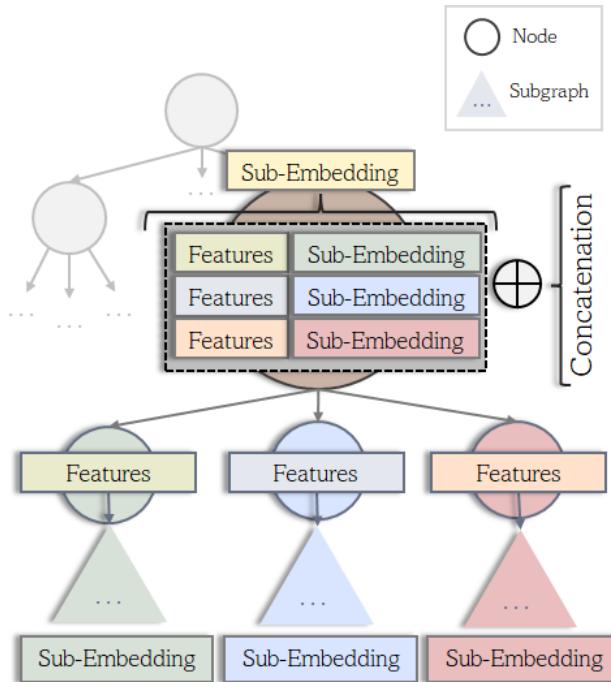


FIGURE 7.4: Visualizing the recursive node encoding scheme based on node features and child nodes.

A proper definition of the embedding function  $f_{\Omega}$  is of great importance for the final effectiveness and utility of the model. As we will show later, we will use a DNN to train and learn an optimal configuration of weights to calculate and determine meaningful representations (Section 7.5).

### 7.2.3 Non-Hierarchical Entity Encoding for Feature Vectors

If we consider Figure 4.5 again, we can recall that not all data entities of interest need to exhibit a hierarchical structure (e.g. data can be an aggregation of many different sources; including non-HDB structures). We will refer to such non-HDB data entities as non-hierarchical entities (NHEs). NHEs, however, can still hold multiple feature values of different types (e.g. categorical, numerical, time-stamps, etc.); and come in various forms (e.g. textual, numeric, etc.). Our focus will now shift to NHEs represented as feature vectors, denoted as FVs (inherently a subset of NHEs).

A simple way to deal with multiple FVs is to concatenate the various features directly, usually after preprocessing, and processing them as a whole; and/or to manipulate FVs further, e.g. generating a lower-dimensional representation like an embedding for the corresponding entity to reduce dimensionality.

FVs typically include a range of different features, especially numerous categorical ones. However, when integrating these features, there may be instances where certain features (e.g. categories) are missing or not directly applicable, resulting in the presence of missing values in the FV. Managing these missing values requires careful consideration. For instance, a simplistic approach like imputing missing values with zeros or the mean value may not accurately represent the data, potentially being incorrect or not being a meaningful interpretation (e.g., calculating the mean of categorical variables is nonsensical).

Therefore, since missing values are very common in our use case, we will adopt a distinct alternative approach by employing the same model architecture approach proposed for the hierarchical trees; albeit in a slightly modified manner. We will interpret and transform the FV (i.e. NHE) entity as/into a hierarchical structure. This approach offers some interesting advantages: Firstly, it ensures framework *consistency* as all embeddings are computed using the same underlying architecture and logic. Secondly, it can leverage a specific type of *contrastive learning* through a graph similarity index/approach. For instance, using the Jaccard coefficient, we would otherwise not be able to calculate tree-node intersections and therefore always result in an overlap of zero. Finally, *missing values* or entries are not an issue, as these represent merely "non-existent" nodes; posing no problem at all and not requiring any other special handling or missing value imputations. This is particularly useful in cases where we have lots of missing categorical and entity-specific data.

Figure 7.5 below shows the transformation of an NHE into a hierarchical equivalence that can function as an input for our model and follows the hierarchical (tree-like) encoding scheme. The new data structure encodes categorical features as child nodes and, therefore, enables us to apply common *graph* similarity metrics (such as the Jaccard coefficient, etc). It employs a trainable look-up table for categorical variables as explicit child nodes and retains numeric features as node attributes. Alternatively, one could also encode the numeric features all as individual child nodes (i.e. with no features in the parent node) as well.

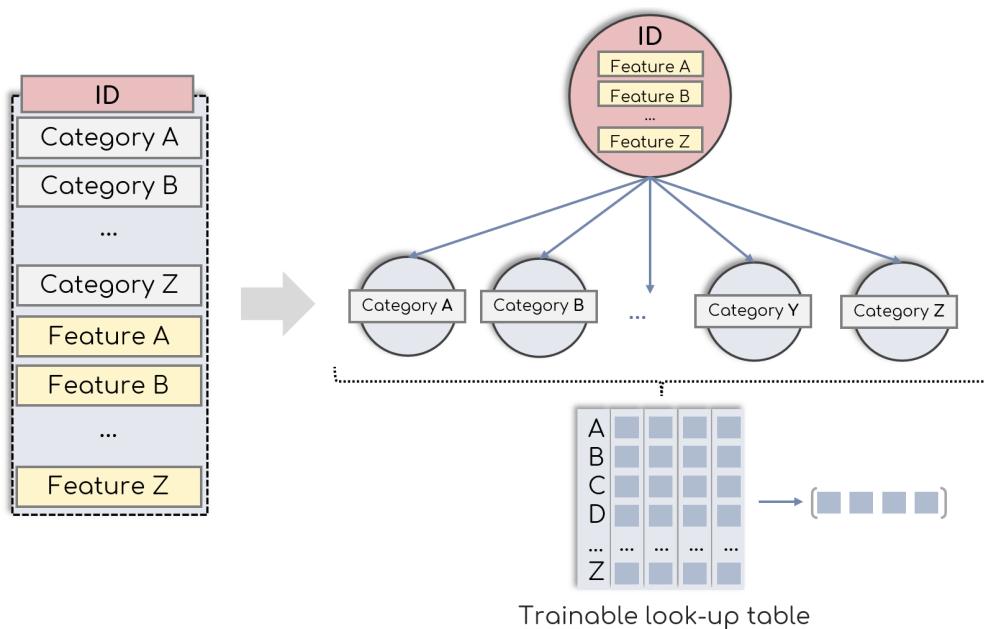


FIGURE 7.5: Depicting the data-structure transformation of non-hierarchical entities. On the left is the raw entity with all its features and categories; on the right is the respective transformed version that can function as input for our model.

### 7.2.4 Composite Graphs: Shared Data Structures for Reducing Memory Overhead & Ensuring Uniqueness

Traditionally, graphs are trained collectively, such as in batches or contrastive methods. The embeddings of individual graph nodes are processed and inferred separately and independently. However, training graphs in isolation has some drawbacks. When using independent graphs, the same node entity may have non-unique representations across different graphs, resulting in inconsistent and confusing (e.g. biased or misleading) downstream analysis. Moreover, duplicate node entries can lead to redundancy and cause memory overhead, especially for larger datasets. E.g., if we have identical sub-graphs in multiple trees, then we would have to store the same data structure multiple times. Note, that duplicate nodes, sub-graphs, etc. must also very importantly have identical features to be considered "duplicates"; otherwise, the node's embedding can not be shared since the node represents different data. That is, duplicates are identical in features and topology.

Therefore, a composite approach that utilizes a *shared* data structure may offer some advantages here. It enables multiple graphs to "point" the same node entity, ensuring that each entity only occurs exactly once, and hence has only one embedding (i.e. a unique node representation). It further reduces memory usage since there are no duplicate nodes; depending on the dataset, this may be significant. Then, during training or inference, however, only nodes from the corresponding graph are active while nodes from other graphs are deactivated. This is important to ensure that the graphs are processed independently. Otherwise, information from other graphs, which are not of interest, can propagate up the model or to the root node. That is, we are effectively only interested in the *shared* embedding/information; we do not want any influence or information from other (sub)-nodes that are not part of the actual graph. Figure 7.6 depicts this idea.

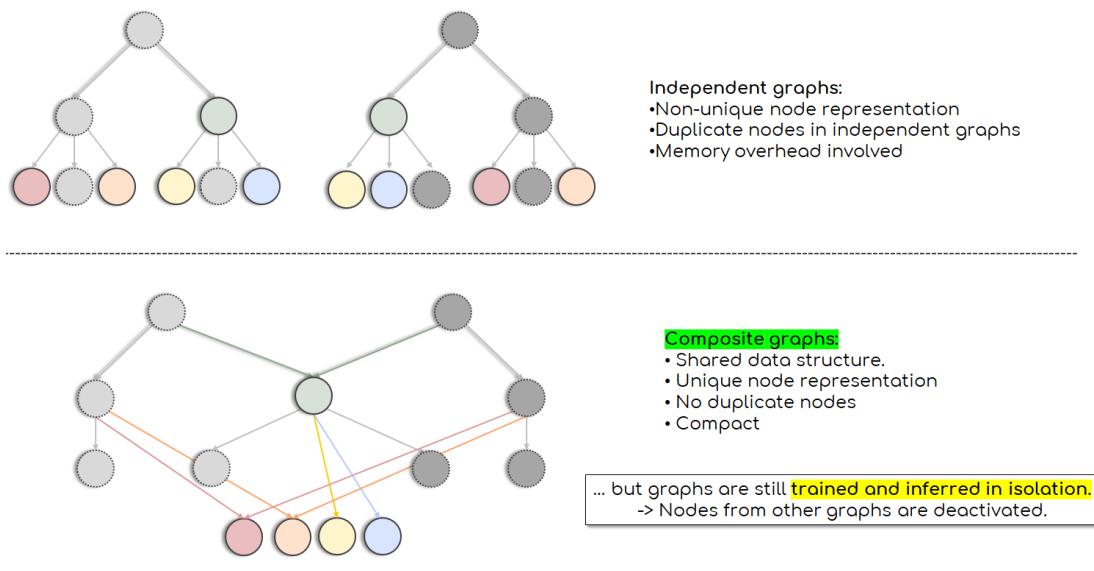


FIGURE 7.6: Illustration of a composite graph approach. Top: traditional method, defining each graph separately. Bottom: merging graphs by connecting nodes with identical features (e.g. for the same entity), ensuring each entity has only one representation. Graphs are trained in isolation (but in context), i.e. considering only nodes contained in the original graph.

### 7.2.5 Encoding Determinism

The encoding transformation from a raw source file (e.g. CSV) to a graph structure is deterministic and unique, including feature normalization and categorical encoding. As a result, the inverse transformation (from the graph back to the original raw CSV file) is deterministic and unique as well. This is very useful since we can now easily compare the original file to the reconstruction to ensure that our transformation is correct (Figure 7.7). In other words, this implies that reversing the encoding process applied to transform data into a graph will yield the (exact) original CSV file. This reversibility can be leveraged and is advantageous for validation, hence, facilitating a straightforward comparison between the initial file and the reconstructed file derived from the graph structure. Validating that the reconstructed CSV file aligns with the original one confirms that the series of transformations used to convert data into a hierarchical tree-based graph structure and subsequently back were correct and accurate, ensuring no loss or corruption of information occurred. Otherwise, on the other hand, non-deterministic mapping between the CSV file and the graph structure would complicate reconstruction, making validation more intricate and requiring storing (i.e. remembering) non-deterministic operations; which would add an additional layer of complexity in this scenario.

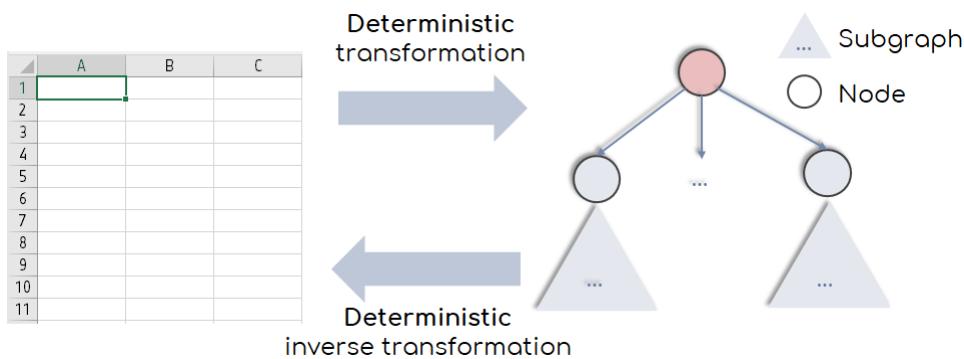


FIGURE 7.7: Determinism and uniqueness of data encoding.

#### Take-Away

We can interpret hierarchical data as trees and express graphs and instances as (a set of) matrices. This enables us to avoid many of the limitations, challenges, and problems that would otherwise be encountered when utilizing a brute-force linearized one-dimensional vector encoding.

#### Keywords

Hierarchical Data • Trees • Matrices • Encoding • Representation Uniqueness

## 7.3 Feature Pre-Processing

Feature scaling and normalization play a central role in ML applications and often decisively influence the success of an ML model (Singh and Singh, 2020). It is therefore important to adequately perform feature pre-processing such as adequate scaling, normalization, or standardization of feature values.

### 7.3.1 Numeric Feature Normalization

**Please note:** Our contribution of "Tanh-Normalizations" has been accepted for publication, after peer review in Guimerà Cuevas and Schmid, 2024b. The Version of Record is available online.

Although many linear and non-linear methods exist (e.g min-max normalization, mean-std standardization, quantile transformers, etc.) we will use and propose a novel scaling method based on the *tanh-estimator* (Hampel et al., 2011) since it has shown to be very robust and highly efficient (Bhanja and Das, 2018; Jain, Nandakumar, and Ross, 2005; Ribaric and Fratric, 2005). Our contribution here was peer-reviewed and accepted for publication.

The *tanh-estimator* (Hampel et al., 2011) is defined as:

$$\phi(x) := \frac{1}{2} \left[ \tanh \left( 0.01 \frac{(x - \mu_\Psi)}{\sigma_\Psi} \right) + 1 \right] \quad (7.4)$$

where  $\mu_\Psi, \sigma_\Psi$  are the mean and standard deviation of the scores of the *Hampel* estimator. Per definition, the range of the *tanh-estimator* is  $[0, 1]$ , but we can constrain it linearly within the range of  $[-1, 1]$  using the normalization of  $\phi'(x) := \tanh(0.01 \frac{(x - \mu_\Psi)}{\sigma_\Psi})$ . One reason for constraining the outputs at an interval with zero mean is that "convergence is usually faster if the average [...] is close to zero" (Le-Cun et al., 2012). This heuristic should also hold for hidden layers, which is also why *Batch-Normalization* (Ioffe and Szegedy, 2015) is often applied to the outputs of activation functions.

Choosing the right *Hampel* parameters, however, is crucial and challenging. A modified tanh-normalization was proposed (Latha and Thangasamy, 2011) that used the raw features' mean and standard deviation rather than the *Hampel* scores. The intrinsic complexity associated with the *Hampel* function was fully removed, resulting in a less complex formula (i.e. less. computation). They did, however, employ a default constant value of 0.01, which we will refer to as the *spread-value* and may still be seen as a configurable hyper-parameter which we denote as  $\alpha$ . The correct choice of  $\alpha$  is still critical since it regulates the spread-value of the output prior to the non-linear *tanh* transformation, hence affecting the effective distributions of the features, which we may still wish to keep close to, for example, a Gaussian distribution. We will propose an adaptive method that directly builds upon the modification as suggested by (Latha and Thangasamy, 2011). The reason for determining the spread factor automatically is to avoid further hyper-parameters and adapt the formula to custom data. We will compute and minimize divergence against a target distribution using the *Wasserstein distance* (WD) (Rüschorf, 1985). Formally, the  $p$ -th WD between two probability distributions  $u, v$  is defined as:

$$WD_p(u, v) := \inf_{\pi \in \Pi(u, v)} \left( \int_{\mathbb{R} \times \mathbb{R}} d(x, y)^p d\pi(x, y) \right)^{1/p} \quad (7.5)$$

where  $\Pi$  is the set of joint distributions  $\pi$  for  $(x \sim u, y \sim v)$  with marginals  $u, v$ . For  $p = 1$  and  $d(x, y) := |x - y|$ , it follows (Ramdas, Trillos, and Cuturi, 2017):

$$WD(u, v) = \inf_{\pi \in \Pi(u, v)} \int_{\mathbb{R} \times \mathbb{R}} |x - y| d\pi(x, y) = \int_{-\infty}^{\infty} |U(t) - V(t)| dt \quad (7.6)$$

where  $U, V$  are the cumulative distribution functions (CDF) of  $u, v$  respectively. Informally, the WD metric may be thought of as the smallest "cost" of converting one

probability distribution to another.

Under the premise that the optimal data distribution of feature values follows a target distribution with desired properties; such as having zero mean, unit variance (LeCun et al., 2012), (e.g a standard Gaussian); the modified normalization proposed by (Latha and Thangasamy, 2011) is sub-optimal due to the constant spread value for all features. Exemplary, we will use a standard Gaussian as our target. Let  $F_Z$  be the standard normal distribution's CDF of the standard Gaussian  $Z$ . Thus,  $f_Z(x) := \frac{e^{-x^2/2}}{\sqrt{2\pi}}$ . Let  $\mu_x, \sigma_x$  be, respectively, the mean and standard deviation of a feature distribution for a specific feature  $x \in \mathbb{R}$ . We replace the fixed spread value with a tunable  $\alpha$ :

$$\phi_X^\alpha(x) := \tanh\left(\alpha \cdot \frac{x - \mu_x}{\sigma_x}\right) \quad (7.7)$$

The goal is to determine the best  $\alpha$  for each feature such that the output distribution of  $\phi_X^\alpha$  follows our (desired) target distribution.  $F_{\phi_X^\alpha}(x)$  be the CDF of  $\phi_X^\alpha(x)$ :

$$F_{\phi_X^\alpha}(x) := \int_{-\infty}^x \phi_X^\alpha(t) dt \quad (7.8)$$

We define the objective loss  $\mathcal{L}_{WD}$  per feature using Equation 7.6:

$$\mathcal{L}_{WD}(\alpha) := \int_{-\infty}^{\infty} |F_Z(t) - F_{\phi_X^\alpha}(t)| dt \quad (7.9)$$

The normalized feature values are constrained to the range  $[-1, 1]$ , whereas the Gaussian is defined for all real values. One may define the KDE for values  $< -1$  as zero and  $> 1$  as one, to allow a direct comparison; or use a truncated Gaussian at a quantile  $q \in ]0, 1[$ , scaled to a domain of  $[-1, 1]$ . Let  $ppf$  be the percent point function (alias. quantile function) of  $f_Z$  such that  $F_Z(ppf(q)) = q$ .

Then, for  $x \in [ppf(q), ppf(1 - q)]$ ,  $f_Z^q(x)$  defines the  $q$ -truncated Gaussian. Low  $q$  values have greater min/max values, which squish the central mass towards the center, whereas high values distribute the mass more towards the tails. The parameter  $q$ , thus, determines the density coverage of the truncation. By varying the standard deviation of the truncated Gaussian, we can choose distributions where the probability mass is either concentrated in the middle or distributed more evenly. To assure equivalent domains (i.e.  $[-1, 1]$ ), we define  $\hat{f}_Z^q(x) := 2 \cdot \frac{f_Z^q(x) - \min(f_Z^q)}{\max(f_Z^q)} - 1$ . Let  $\hat{F}_Z^q(t)$  be the respective KDE function. The new truncated loss then is:

$$\mathcal{L}_W^q(\alpha) := \int_{-1}^1 |\hat{F}_Z^q(t) - F_{\phi_X^\alpha}(t)| dt \quad (7.10)$$

Direct minimization of  $\mathcal{L}_W^q$  over  $\alpha$  is not convex, as can be seen in Figure 9.19, where convexity is broken for several different feature distributions around  $\alpha \approx 0.6$ . However, when a normal Gaussian is used as the target distribution (and also for several other distributions), we observe that the loss function  $\mathcal{L}_W^q(\alpha)$  is quasi-convex (Figure 9.19), which implies a single (global) minimum. Thus, to calculate the optimal spread per feature given  $q$ , the efficient numerical optimization method *Brent*<sup>3</sup> (Brent, 1971) can be leveraged. Alternatively, other gradient-free algorithms (Brent, 2013) may also be used, but because this pre-processing step will only be

<sup>3</sup>E.g. Brent uses a fast-converging secant technique or inverse quadratic interpolation, but if required, adapts a more robust bisection approach.

executed once, performance here is not critical. The optimum spread value  $\dot{\alpha}$  that minimizes  $\mathcal{L}_W^q(\alpha)$  is:

### 7.3.2 Ideal Spread Factor

The *ideal* spread values (Eq.7.7) are incorporated into the MTN formula (Latha and Thangasamy, 2011), wherein the static spread value of e.g. 0.01 is replaced by the corresponding ideal  $\dot{\alpha}$  of each feature. Consequently, the ideal transformation function for each individual feature becomes:

$$h_{\dot{\alpha}}(x) := \tanh\left(\frac{\dot{\alpha}(x - \mu_X)}{\sigma_X}\right) \quad (7.11)$$

### Trainable Spread Factor

Once  $\dot{\alpha}$  has been pre-computed independently for each feature, it may be left constant/fixed and does not need to be altered. In many machine learning domains, one can here additionally learn a training-objective-specific spread value  $\gamma$  (to fine-tune  $\dot{\alpha}$ ) as part of the training process, e.g. via back-propagation optimization. Let  $\tau > 0$  be a trainable parameter. Then, a corresponding *trainable* tanh-estimator  $h_\gamma$  can be defined by simply setting  $\gamma := \tau\dot{\alpha}$ , and so,  $h_\gamma(x) := \tanh\left(\frac{\tau\dot{\alpha}(x - \mu_X)}{\sigma_X}\right)$ . By adjusting  $\tau$ , we can learn the ideal spread of the feature distribution during training end-to-end. Because the backward optimization here is solely dependent on  $\tau$ ,  $\dot{\alpha}$  is still only computed once before the training and can be reused for different training runs. The model can now "learn" which normalization (i.e. spread factor) is the best for optimizing a specific model loss and training objective; also in conjunction with other features. Recall that otherwise  $\dot{\alpha}$  is determined independently of other features and specific training goals.

### Spread Value

To summarize the role of the spread value: the parameter  $\alpha$  in the feature normalization plays a crucial role, as extreme values can disrupt the normalization process. If  $\alpha$  is too large, most values get close to one, while a too-low value drives them toward zero, counteracting the actual normalization goal. Determining the optimal  $\alpha$  relies on the desired distribution; and can e.g. be visualized through the KDE graph for a *tanh*-normalized Gaussian. Standardization seeks zero mean and unit standard deviation without altering the distribution shape. Our proposed optimization process involves minimizing the Wasserstein Distance (WD) loss between the output KDE and the desired distribution. The optimal spread value is denoted as  $\dot{\alpha}$ ; and is pre-computed. Note, how the search for the best  $\alpha$  involves minimizing the WD loss rather than WD itself (we are constrained to only adjust the spread value as the independent variable; we cannot otherwise modify the distribution of features). Optionally, we can learn a further parameter  $\tau$ , and use  $\tau\dot{\alpha}$  as the final effective spread; where  $\tau$  is optimized to minimize an objective loss (not the WD anymore).

The ultimate value of  $\alpha$  significantly influences the *tanh* normalization's KDE and, subsequently, the WD. It also affects the range and quantile distribution of the transformed features. Overall, using a standard spread of 0.01 is suboptimal, and relying on random guesses or manual adjustments is neither practical nor sensible. An automatic mathematically motivated method is therefore needed (which we term Wasserstein Tanh-Normalization).

### 7.3.3 Time-Stamp Encoding

Time can be measured *cumulatively* or *cyclically*. Cumulative time characteristics measure the total time elapsed since a given starting point. However, we are often interested in finding correlations and patterns between cyclic time properties (e.g. 24-hour day cycles, 12-month year cycles, etc.). In such a case, using a cumulative representation would be sub-optimal.

Yet, when dealing with time-stamps in the format of YEAR-MONTH-DAY and HOUR:MIN:SEC (e.g. 2014-09-17 12:12:26) we are interested in both encoding strategies, cumulative for "years" and cyclical for the rest. As for the cumulative part, we can simply accept the raw score, whereas, for the cyclical part, we can use the conventional encoding approach of applying periodic functions, similar to the concept of complex numbers. In fact, we can map each time feature  $x_t$  (e.g. hour) - normalized by the maximum of its domain  $x_t^{\max}$  (e.g. 24 hours) - respectively onto the complex unit-circle with:

$$f_t(x_t) := \cos\left(2\pi \frac{x}{x_t^{\max}}\right) + i \sin\left(2\pi \frac{x}{x_t^{\max}}\right) \quad (7.12)$$

where  $i$  is the complex imaginary number. Note that the feature "days" is cyclic but a month can vary in length. One might use the mean length of the synodic month, which is approximately 29.53059 days, and cap respectively any greater values. A better method, however, is to instead encode the day of the year as a value between [1,365] for regular years (neglecting leap years of 366 days); and map this on the unit-circle.

### 7.3.4 Ordinal Feature Encoding

Some numerical features have an ordinal structure, e.g. the software version for a specific prototype. However, it cannot be guaranteed that after the data set has been divided into train and test sets, the train set will contain all of the versions. This can be problematic when trying to derive an appropriate encoding of those versions (e.g. in the test set or when predicting new data), since they were not included in the train set and, therefore no direct ordinal equivalent exists.

For example, suppose the train set has an explicit mapping of versions to values  $\{v.0 : 0, v.1 : 1, v.2 : 2\}$ . If a new version  $v.1.15$  needed to be encoded, there would not exist any corresponding matching version. In other words, we can not directly expect the ordinal encoding of the test set to work on previously unseen data.

One possible intuitive way of solving this is to map the version to its closest neighbor (i.e. version) and assign its encoding. Following the example above, versions  $v.0.4, v.1.15, v.1.7, v.2.3$  would be respectively mapped as  $\{v.0.4 : 0, v.1.15 : 1, v.1.7 : 2, v.2.3 : 2\}$ . This can be seen as a hierarchical order-preserving encoding approach; where unseen/non-matching instances are mapped to their closest hierarchical index position.

#### Take-Away

Proper pre-processing and scaling of feature values is critical for effective training. Different data types necessitate different processing methods. For numerical features, we recommend using an adaptive outlier resilient normalization technique that automatically adjusts to the data distribution.

### Keywords

Pre-processing • Scaling • Feature Values • Effective Training • Adaptive Outlier Resilient Normalization

## 7.4 Tree-Data Augmentation

By using hierarchical graphs, we have the ability to effectively expand the size of the dataset synthetically by generating new artificial graph instances through the process of *augmentation*. There are numerous approaches and possible ways to data augmentation, but we will be adopting a straightforward "merge & sample" technique (Figure 7.8). Initially, we select a random subset of  $k$  graphs from the complete pool of graphs  $\Theta \subseteq \Omega$ , based on a fixed probability value  $p$ , and merge them together into a unified tree. Subsequently, we generate a random sub-graph from this newly formed union tree. Furthermore, this can be repeated several times to create as many new synthetic hierarchical trees as required. Additionally, selecting different values for  $k$  or sub-sampling on multiple multiple times are also viable options.

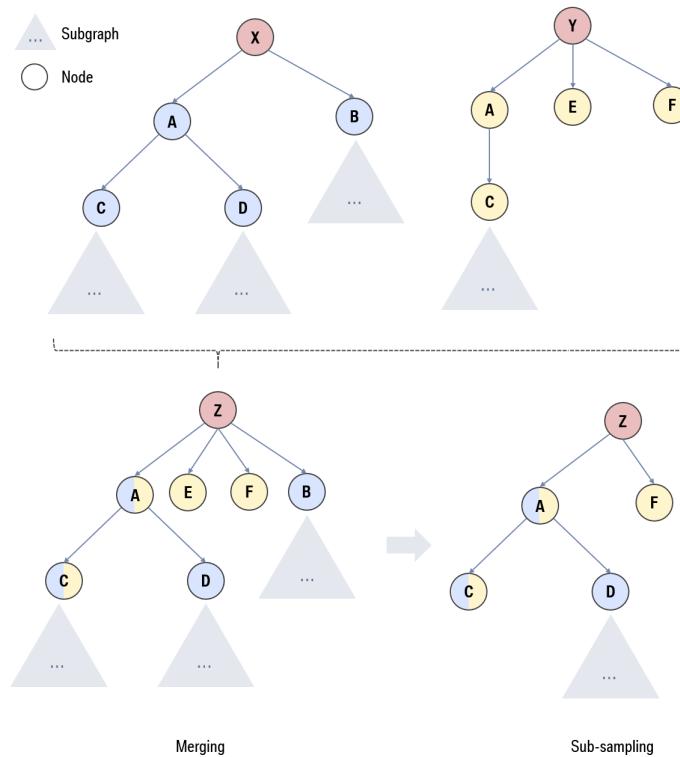


FIGURE 7.8: Synthetically generating new graphs from given graphs (in this case out from a total of two graphs).

If  $p^{-1} := |\Theta|$ , then we sample - and consequently augment the data - based on a *uniform distribution*; inversely to the number of instances. However, by doing so we would have problems when dealing with imbalanced (e.g. biased) data sets where given groups or subgroups exist more often than other hierarchical graph structures. In fact, we would be over-sampling those structures and under-sampling rarer ones and, thus, have a bias in our augmentation. To counteract this, we can use conditional sampling that samples graphs based on their *rareness*. A straightforward

question here is how to define the "rareness" of a given graph, but we can here e.g. simply use the average graph similarity - e.g. *Jaccard-Coefficient* (JC) - against other graphs; or more advanced approaches like determining similarities (or conducting clustering) in the latent embedding/manifold space. This is based on the motivation that rare graphs are those that have low similarities to others, whereas graphs with frequently occurring structures and patterns yield higher average similarity scores and are, thus, less of interest as they are less novel (i.e. special).

As an example, let us again consider the JC score for two graphs - which essentially measures the overlap of graphs relative to their total sizes - and is defined as<sup>4</sup>:

$$JC(\omega, \omega') := \frac{|\Omega_\omega^\Delta \cap \Omega_{\omega'}^\Delta|}{|\Omega_\omega^\Delta \cup \Omega_{\omega'}^\Delta|} = \frac{|\Omega_\omega^\Delta \cap \Omega_{\omega'}^\Delta|}{|\Omega_\omega^\Delta| + |\Omega_{\omega'}^\Delta| - |\Omega_\omega^\Delta \cap \Omega_{\omega'}^\Delta|}. \quad (7.13)$$

Then, given a graph instance, we can define the total similarity score for an instance  $\omega \in \Omega$  by using JC-based similarity scores e.g. on either sub-graph similarities or full-graph similarities.

$$\begin{aligned} \text{JC-subgraph-score}(\omega) &= \sum_{\omega'}^{\Omega \setminus \Omega_\omega^\Delta} JC(\omega, \omega') \\ \text{JC-graph-score}(\theta) &= \sum_{\theta'}^{\Theta \setminus \{\theta\}} JC(\theta, \theta') \end{aligned} \quad (7.14)$$

The JC-graph-score calculates the JC score for entire graphs, while the JC-subgraph-score calculates it for every possible sub-graph. Computing the JC-subgraph score is, thus, extremely computationally expensive and absolutely impractical. However, it can still be used in conjunction with a Monte Carlo score approximation. Still, we suggest using the JC-graph-score instead; it is much more computationally feasible.

The full JC score matrix can be retrieved by generating (e.g. pre-computing) the pairwise symmetric Jaccard similarity matrix once and then summing up or averaging the respective rows (or columns; as the matrix is symmetric). However, computing such a JC matrix scales quadratically with the number of graphs and might become somewhat computationally expensive for very large datasets. However, since it is symmetric; we only need to compute the upper/lower triangle matrix of it:

$$\vec{v} := \mathbb{E}_{dim=1} \begin{bmatrix} 1 & JD(\theta_0, \theta_1) & JD(\theta_0, \theta_2) & \cdots & JD(\theta_0, \theta_{|\Theta|-1}) \\ JD(\theta_1, \theta_0) & 1 & \cdots & \cdots & \cdots \\ JD(\theta_2, \theta_0) & & 1 & \cdots & \cdots \\ \cdots & \cdots & & 1 & \cdots \\ JD(\theta_{|\Theta|-1}, \theta_0) & \cdots & \cdots & & 1 \end{bmatrix} \triangleq \begin{bmatrix} v_{\theta_0} \\ v_{\theta_1} \\ v_{\theta_2} \\ \vdots \\ v_{\theta_{|\Theta|-1}} \end{bmatrix} \quad (7.15)$$

Overall, given average similarity scores  $\vec{v} \in [0, 1]^{|\Theta| \times |\Theta|}$  of all graphs  $\theta \in \Theta$ , we can then easily build an inversely related score distribution:

$$p_\theta = 1 - \frac{v_\theta}{\sum_{\hat{\theta}}^{\Theta} v_{\hat{\theta}}} \quad (7.16)$$

Note, that  $v_\theta \in [0, 1]$ , and we subtract the normalized probability from one as we are interested in sampling *uncommon* tree structures as well to avoid generating

<sup>4</sup>Recall,  $\Omega_\omega^\Delta$  is the set of all nodes within a given sub-tree with root  $\omega$  (including  $\omega$ ).

biased structural overlaps, and so prevent over-sampling; i.e. we want to also sample those structures which have low similarities to all other trees and, thus, occur less frequent. In other words: we want our data augmentation to be balanced and not biased. Moreover, one could also apply an (inverse) Soft-Max probability transformation, e.g. to avoid probabilities of zero; or also simply just use the standard *inverse proportional* probability score transformation.

### Take-Away

Data augmentation can help to improve training by creating additional and new (synthetic) data. In our case, we can directly **produce new data by merging and combining numerous different trees** at random. Typically, we aim to avoid over-sampling common patterns and under-sampling less frequent (i.e., minority) ones: the synthetic data should *not* be biased.

### Keywords

Data Augmentation • Synthetic Generation • Merging • Combining Graphs

## 7.5 Self-Supervised Contrastive Representation Learning on Tree Structures

We are interested in learning meaningful representations for a set of (hierarchical) tree structures so that they can be easily used in other ML models or for tasks such as clustering, object similarity detection, outlier identification, or other subsequent downstream processing. To achieve this, we will first employ a self-supervised-based approach by determining and extracting similarity properties from the available dataset. Then, we will train a model in a semi-supervised manner to learn to embed similar trees close to each other, while ensuring that dissimilar trees are far apart. In other words, the closer two trees are in terms of similarity, the closer they should be in the embedding space. Conversely, the more dissimilar two trees are, the farther apart they should be.

We measure the similarity between two hierarchical trees  $\omega_{\{i,j\}} \in \Theta$  in the dataset using a similarity score function denoted as  $\simeq (\omega_i, \omega_j)$ . It is important to note that we can use any desired similarity function, such as graph-based, embedding, or kernel-based distance metrics. It is worth mentioning that the similarity relation is not an equivalence relation because it is not necessarily transitive, i.e.,

$$\omega_0 \simeq \omega_1 \wedge \omega_1 \simeq \omega_2 \not\Rightarrow \omega_0 \simeq \omega_2 \quad (7.17)$$

For instance, the Jaccard coefficient does *not* satisfy the triangle inequality. Figure 7.9 below illustrates this exemplary for a pair of three hierarchically structured trees. Thus, when calculating the embeddings, we must make sure that the embeddings of the pair  $(\omega_0, \omega_1)$  and  $(\omega_1, \omega_2)$  are close in the embedding space, but not the pair  $(\omega_0, \omega_2)$ .

Additionally, a similarity score function may not satisfy the properties of a distance metric, as the triangle inequality may not hold. That is, when choosing a similarity function  $\simeq (\omega_i, \omega_j)$  one has to pay special attention to whether or not the

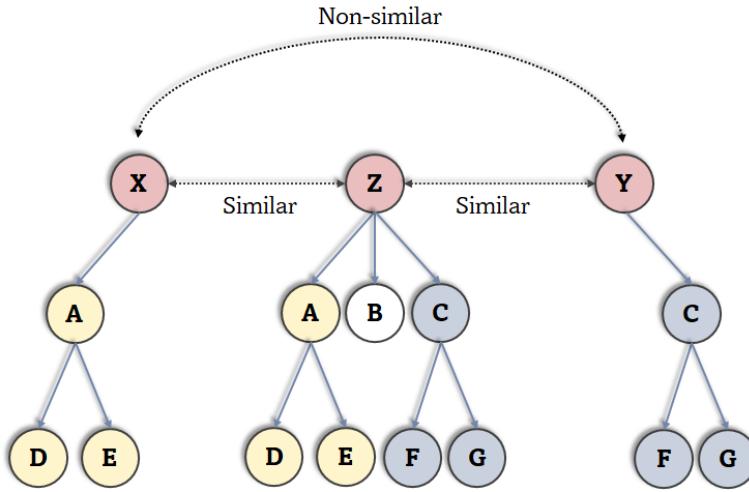


FIGURE 7.9: Illustration of the non-transitiveness of tree structures over the Jaccard similarity relation.

function itself defines a *distance* metric  $d$ . For a given function  $d(\omega_i, \omega_j)$  to be considered as such, the four criteria positivity, identity, symmetry, and triangle inequality must always hold:

1. **Positivity:**  $d(\omega_i, \omega_j) \geq 0$
2. **Identity:**  $d(\omega_i, \omega_j) = 0 \Leftrightarrow \omega_i = \omega_j$
3. **Symmetry:**  $d(\omega_i, \omega_j) = d(\omega_j, \omega_i)$
4. **Triangle inequality:**  $d(\omega_i, \omega_k) \leq d(\omega_i, \omega_j) + d(\omega_j, \omega_k)$

Overall, one can then make certain assumptions or not, depending on whether it is a metric or not. Again, if we take JC as an example, we can form the Jaccard distance (JD) metric via  $JD(\omega_0, \omega_1) := 1 - JC(\omega_0, \omega_1) = \frac{|\Omega_{\omega_0}^\Delta \cup \Omega_{\omega_1}^\Delta| - |\Omega_{\omega_0}^\Delta \cap \Omega_{\omega_1}^\Delta|}{|\Omega_{\omega_0}^\Delta \cup \Omega_{\omega_1}^\Delta|}$ . The fact that JD is a proper metric on finite collections was proven by Kosub, 2019.

An important distinction needs to be made between two types of hierarchies: strict and non-strict. In strict hierarchies, nodes can only overlap if they share the same parents from that node to the root node. For example, in cases of part-groups and sub-parts, a part can only overlap with a different part if they belong to the same sub/parent group. On the other hand, non-strict hierarchies do not consider the parent path-nodes and only focus on the actual node itself. A good example of this is environmental conditions, where it may or may not have been recorded for specific (distinct) parts. Thus, in strict hierarchies, similarity is determined and assessed based on the parent-path nodes, whereas in non-strict hierarchies, the focus is solely on the overlaps between nodes, regardless of their parent nodes.

Furthermore, it is worth noting that nodes can also contain features. Therefore, when comparing graphs, it is possible for a graph with a lower Jaccard coefficient to be considered more "similar" if the feature values within each overlapping node are more alike compared to a graph with a higher Jaccard score but significantly different feature values. Hence, relying solely on the Jaccard coefficient as a metric is inadequate (i.e. poor, insufficient) for accurately defining similarity. However, it

does offer useful initial approximations and provides a general understanding of the concept of similarity.

We therefore propose and adopt a recursive DL architecture that generates representative embeddings for each of the active sub-trees in the hierarchy. This approach follows a bottom-up strategy; beginning from the leaf nodes and propagating hierarchical information upwards towards the root node. This generates a final root-representation vector, representing the entire graph, which is then used to establish similarities, detect clusters, and identify outliers.

### Take-Away

Determining similarity on graphs in an unsupervised setting is a challenging task due to the **lack of clear definition and interpretation of the term "similarity."** Similarity can be defined based on structural/topological similarity, similarity in feature values, or based on both.

### Keywords

Unsupervised Setting • Graph Similarity • Unclear Definition • Interpretation • Structure Similarity

#### 7.5.1 Loss Function

Defining a sensible and accurate loss function is of great importance as it serves as the core foundation for our ML optimization. Failing to establish a "good" loss or training objective will result in our model optimizing its weights based on a sub-optimal loss function. We must, thus, not overlook or underestimate the significance of the loss function as it directly influences and shapes the quality of the embeddings; and thus the performance and effectiveness of further downstream tasks. That is, training (graph) embeddings over a poor/bad loss will inevitably lead to poor/inadequate embedding representations. To ensure quality and counteract this, we incorporate five essential properties into our loss function:

1. **Reconstruction:** How accurately can we recreate the original input (graph, sub-tree, etc.) back from our embedding representation?
2. **Composition:** Does the embedding capture the independent nature of the child nodes while still representing the entire graph composition as a whole? Are the features within the embeddings uncorrelated with each other?
3. **Similarity:** Do the pairwise similarities between embeddings align with the similarity/distance function, such as  $\simeq(\omega_i, \omega_j)$  or  $d(\omega_i, \omega_j)$ ?
4. **Contrast:** Are similar embeddings close together and dissimilar embeddings far apart in the embedding space? Does the embedding space maintain the neighborhood of similar graphs?
5. **Classification:** Is it possible to predict and determine the class label directly from the embeddings accurately?

Note, that *contrast* may be considered as an implication of *similarity*, i.e. if the similarity loss is perfect, then one might argue that so is contrast loss as a consequence. However, optimizing the similarity directly might be too complex/difficult

to achieve and, moreover, might also not have the same relevance. For example, sometimes our ultimate goal may be to have good "contrasting" representations, which might entail compromising on encoding local similarities accurately. So instead of accurately wanting to represent the degree of similarity among similar objects, we might just want to focus effectively on differentiating/classifying between "different" instances. So overall, capturing the exact level of similarity between objects can be challenging, but distinguishing between similar and dissimilar objects can be relatively much easier and also more relevant. To achieve this, we divide the loss into multiple components and assign separate weights to each component based on our preferences. This approach grants us more precise control over the embedding generation/learning process.

Similar applies to *composition* and *reconstruction*, where composition could be regarded as an implication of the reconstruction quality.

### Reconstruction- & Composition Loss: Auto-Encoders

Reconstruction losses can typically be modeled using Auto-Encoders (AE) where the goal is to reconstruct or approximate the original input out of its latent space (embedding). Unlike typical AEs, our input is not in tabular form, but instead given as a hierarchical graph, which we map, as explained earlier, into matrices (Figure 7.10). The idea is that if the embedding perfectly represents and encodes the input data and no information was lost, distorted, corrupted, etc., then one should be able to (in theory) perfectly reconstruct the original input from it. And so, the more accurately we can approximate this reconstruction, the higher the quality of our representation or embedding vector can be assumed to be.

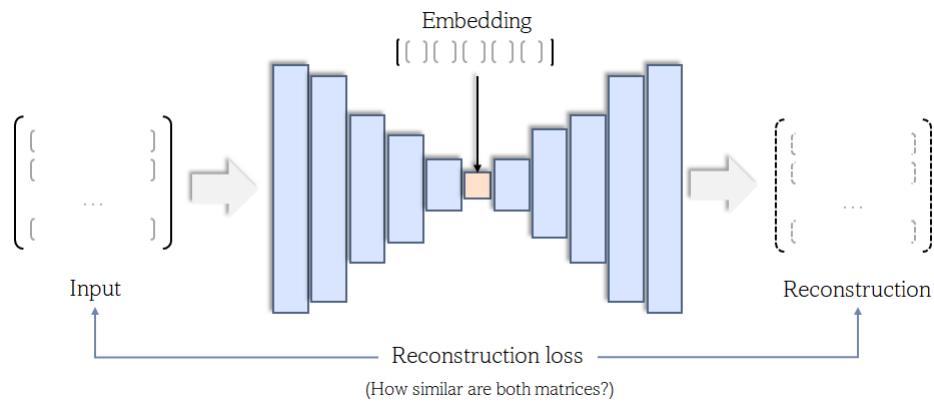


FIGURE 7.10: Illustration of the concept of reconstruction losses using Auto-Encoders (AE). Here, an input matrix is fed into an AE and its reconstruction output is compared against the original input.

Let  $U$  be an input matrix, and  $\hat{U}$  its corresponding reconstruction. The reconstruction loss can be calculated by measuring the difference of pairwise matrix-component values, e.g. by using the mean squared error (MSE):

$$\text{reconstruction loss} \equiv \text{loss}_{\text{rec}} := \mathbb{E}[(U - \hat{U})^{\circ 2}] \quad (7.18)$$

where  $[.]^{\circ [.]}$  is the *Hadamard power*. For the composition loss, we try to enforce a zero correlation between distinct row entries (child) of the original and reconstructed matrix, while simultaneously assuring perfect correlation between equivalent rows:

$$\text{composition loss} \equiv loss_{comp} := \mathbb{E}[(Cos(U, \hat{U}) - \mathbf{I})^{\circ 2}] \quad (7.19)$$

where  $Cos(U, \hat{U})$  denotes the respective cosine similarity matrix:

$$Cos(U, \hat{U}) := \frac{U\hat{U}^T}{\|U\|_2\|\hat{U}\|_2} \quad (7.20)$$

and  $\mathbf{I}$  the identity matrix with ones in the diagonal and zeros elsewhere.

### Similarity- & Contrast Loss: Embeddings

We want similar trees to yield similar embeddings. We will train our models by ensuring these properties hold for any random-batch combination of data instances (i.e. trees). Let  $\mathbf{Y}$  be the symmetric similarity/distance matrix with ones along the diagonal for any arbitrary but fixed batch (e.g. the pairwise Jaccard matrix). We can compute  $\mathbf{Y}$  during runtime for each batch individually, or use dynamic programming and pre-computation to efficiently retrieve entries of the matrix  $\mathbf{Y}$ . Alternatively, we could also use a label matrix, with components one if the label matches and zero otherwise. Let  $Z$  be the embedding matrix of  $U$ . We then define the similarity loss as:

$$\text{similarity loss} \equiv loss_{sim} := \frac{(Corr(Z, Z) - \mathbf{Y})^{\circ 2}}{(\max[1 - \mathbf{Y}, \mathbf{Y}])^{\circ 2}} \quad (7.21)$$

where, again,  $\circ$  denotes here the *Hadamard power*,  $\max$  the element-wise *max*-operation, and  $Corr(Z, Z)$  refers to the Pearson correlation matrix, defined as:

$$Corr(A, B) := \frac{\mathbb{E}[(A - \mu_A)(B - \mu_B)^T]}{\sigma_A \sigma_B} \quad (7.22)$$

with  $\mu, \sigma$  respectively being the mean and the standard deviation. Essentially, we are calculating the error ratio against the maximum possible squared similarity error and constraining the loss into a range between zero and one. It is also possible to omit the denominator and simply regard the numerator as the loss. One could also use the cosine similarity here instead; but notice how *when* the mean is zero the cosine similarity yields the same results as the Pearson correlation. In general, correlation and cosine similarities have different interpretations. Overall, we chose correlation to also better enforce embedding representations with linear relationships.

We then further define the pairwise embedding similarity score on trees based on the correlation between their respective embeddings:

$$ES(\omega, \omega') := |1 - Corr(f_\Omega(\omega), f_\Omega(\omega'))| = |1 - Corr(z, z')| \quad (7.23)$$

where, again,  $f_\Omega(\omega) = z$  is the embedding function for any  $\omega \in \Omega$ . As we trained the correlation to be either zero or one, we can expect  $Corr(z, z')$  to return values in the range of  $\approx [0, 1]$ . Yet, to actually guarantee values between zero and one, we can define  $\hat{ES}(\omega, \omega')$  which scales or bounds the values accordingly. Thus, with  $\hat{ES}(\omega, \omega') \in [0, 1]$ , an embedding similarity of 0 indicates a perfect (positive) correlation, i.e. similarity, whereas a value of one denotes total non-correlation, i.e. maximum dissimilarity.

The implementation of the contrast loss, however, is slightly more intricate. Let  $\vec{v}_\omega$  represent a set of pairwise similarities for a specific instance  $\omega$ . For instance, if

we were using the Jaccard distance,  $\vec{v}_{JD,\omega}$  would be the Jaccard similarity vector for a given node  $\omega$ , calculated as  $\vec{v}_{JD,\omega} := [JD(\omega, \omega_0), JD(\omega, \omega_1), \dots, JD(\omega, \omega_{|\Theta|})]$ . Similarly, if we were using embedding similarity,  $\vec{v}_{ES,\omega}$  would be the corresponding embedding similarity vector  $\vec{v}_{ES,\omega} := [1 - \hat{ES}(\omega, \omega_0), 1 - \hat{ES}(\omega, \omega_1), \dots, 1 - \hat{ES}(\omega, \omega_{|\Theta|})]$ .

To generalize, let  $\vec{v}_{\simeq,\omega} := [1 - \simeq(\omega, \omega_0), 1 - \simeq(\omega, \omega_1), \dots, 1 - \simeq(\omega, \omega_{|\Theta|})]$  be the similarity array for a given similarity (or distance) function  $\simeq(\dots)$ . Let further  $\vec{arg}_{\simeq,\omega}$  be the vector corresponding to the arg-sort of  $\vec{v}_{\simeq,\omega}$  for a given node  $\omega$ :

$$\vec{arg}_{\simeq,\omega} := \text{argsort}[\vec{v}_{\simeq,\omega}] = \text{argsort}\left[\left(\simeq(\omega, \omega_0), \simeq(\omega, \omega_1), \dots, \simeq(\omega, \omega_{|\Theta|})\right)\right] \quad (7.24)$$

and so, *argsort* returns the indices that would sort a given vector in descending order (not unique for vectors containing duplicate values). We define the inverse binary operation *gather* that reorders a given vector  $\vec{v}$  based on a sequence of indices, such that the actual sorting operating  $\text{sort}[\vec{v}] \triangleq \text{gather}[\text{argsort}[\vec{v}], \vec{v}]$ .

Let  $\vec{v}_{Y,\theta}$  denote our target/ground truth reference similarity array for  $\theta$ . We then define the resemblance error  $\epsilon$  as a measure of how well the distance ordering is respected:

$$\epsilon_{res} := \sum_{\theta \in \Theta} (\text{sort}[\vec{v}_{\simeq,\theta}] - \text{gather}[\vec{arg}_{Y,\theta}, \vec{v}_{\simeq,\theta}])^2 \quad (7.25)$$

Simultaneously, we want to maximize the contrast between similar and dissimilar embeddings. We can achieve this using a similar logic, but instead subtracting the reversed (flipped) sorted embedding distances. We, therefore, define the flipped resemblance error as:

$$\epsilon_{flip} := \sum_{\theta \in \Theta} (\text{sort}[\vec{v}_{\simeq,\theta}] - \text{gather}[\text{reverse}(\vec{arg}_{Y,\theta}), \vec{v}_{\simeq,\theta}])^2 \quad (7.26)$$

This allows us to define the overall contrast loss as:

$$\text{Contrast loss} \equiv \text{loss}_{con}^A := \frac{\epsilon_{res}}{\epsilon_{flip} + \lambda} \quad (7.27)$$

for a small  $\lambda$  to prevent the edge-case of zero division. Overall, we are trying to minimize the loss by either minimizing the resemblance error or maximizing the contrast. Also, instead of using the Euclidean distance as contrast, we can also use the sum of the correlations of all corresponding embedding pairs (similar to before):

$$\begin{aligned} \text{loss}_{con}^B := & - \sum_{\theta \in \Theta} \text{Corr}(\text{sort}[\vec{v}_{\simeq,\theta}], \text{gather}[\vec{arg}_{Y,\theta}, \vec{v}_{\simeq,\theta}])^2 \\ & + \sum_{\theta \in \Theta} \text{Corr}(\text{sort}[\vec{v}_{\simeq,\theta}], \text{gather}[\text{reverse}(\vec{arg}_{Y,\theta}), \vec{v}_{\simeq,\theta}])^2 \end{aligned} \quad (7.28)$$

Moreover, we also propose a further alternative exponential contrast loss: Let  $D$  represent the Euclidean distance matrix between each pair of embeddings. As  $D$  is symmetric, we focus on its lower triangular matrix  $L_D$ . Likewise, let  $L_Y$  be the lower triangular matrix of  $Y$  (as it is also symmetric). The exponential contrast loss is then calculated as follows:

$$\text{Contrast loss} \equiv \text{loss}_{con}^C := \sum (e^{\circ - L_D} - L_Y)^{\circ 2} \quad (7.29)$$

By integrating the exponential function, smaller differences between elements of  $L_D$  and  $L_Y$  are even further amplified, resulting in a larger contrast loss. This

amplification, thus, helps to increase the discriminative nature of the loss function; and focuses more on optimizing for more "severe" errors.

Also, note how exact pairwise full ground truth similarity scores or metrics are not crucially necessary. Instead, instances can also be deemed similar if they share the same label (e.g. classification label) and different otherwise; as already mentioned earlier. I.e., we can also just work with labels that indicate either similarity or dissimilarity, and we do not necessarily need the exact *degree* of similarity.

However, the formulas mentioned above are only applicable if there is a defined reference similarity score array for every pair of graph instances (i.e. if we know  $Y$ ). However, even without any ground truths (unsupervised learning), we can still perform contrastive learning in a self-supervised manner. One approach is to augment and manipulate the graph and assign positive similarities to different augmented versions of the same graph while setting similarities between different graphs to zero. Another approach is to pretend that in every batch, half of the graphs are similar and half are not (randomly chosen each time). By repeatedly doing this, we aim for that similar objects will be easier to cluster together and dissimilar objects easier to tear apart.

### Classification Loss

Given instances with corresponding class labels, we can compute and optimize the classification loss on the predictions induced by the respective graph embeddings. We (can) use the classification loss in addition to the above-defined losses (Figure 7.11); as part of a multi-loss function. Classification losses are typically calculated using the *cross entropy loss*; defined using pairs of probability predictions  $\hat{y}_i$  with corresponding ground truth labels  $y_i$ :

$$\text{Cross-entropy loss} \equiv \text{loss}_{\text{clas}} := \sum_{i=0}^{\text{dim}(\hat{y})-1} -y_i \log(\hat{y}_i) \quad (7.30)$$

The probability scores for  $\hat{y}_i$  are then obtained using the *Softmax* function, which is defined as:  $\text{softmax}(\hat{y}_i) = \exp(\hat{y}_i) / \sum_{j=0}^{\text{dim}(\hat{y})-1} \exp(\hat{y}_j)$ .

Classification can be performed at the root level, as well as at any other level within the tree. In such cases, it could be beneficial to assign importance weights to the classification loss based on the tree level. This way, the embedding of the root node could e.g. have the highest weight, while the embeddings of the bottom nodes would have progressively smaller weights. This is also motivated by the fact that the root node "sees" the entire tree, and, therefore, can and should provide the most accurate result. In contrast, lower levels of the hierarchy have limited visibility of the data, and are thus more likely to have prediction errors and uncertainties, particularly when crucial information is only available higher up. Consequently, their prediction errors should carry less weight (e.g. during optimization).

Conducting classification at deeper levels can positively contribute to the overall performance and identify the correct classification/label much earlier; it can forward that classification information upwards, and also shape the sub-embeddings accordingly. In fact, optimizing solely on the highest level (i.e. root node) could even be (quite) hard otherwise: principles of algorithmic "divide-and-conquer".

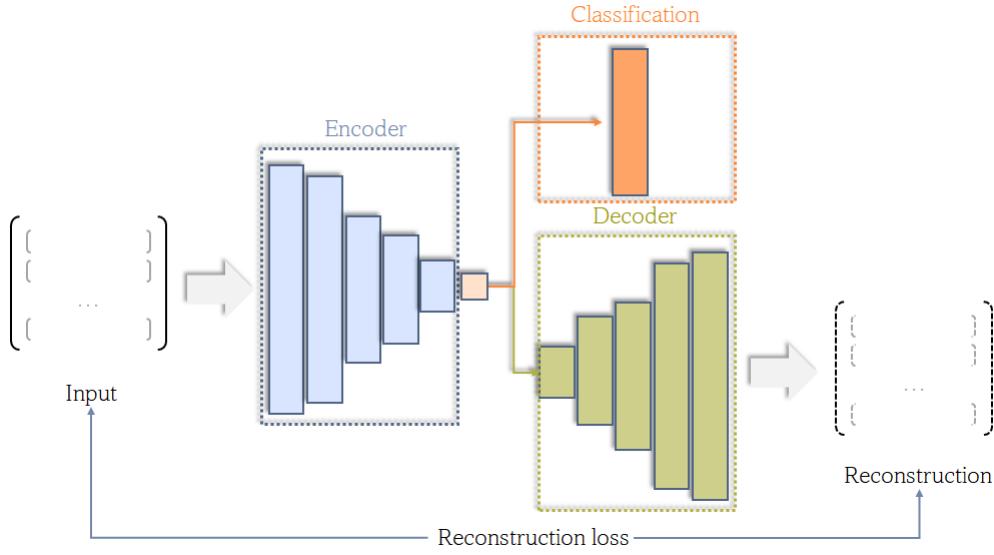


FIGURE 7.11: Enhancing the model to support classification losses.

### Total Loss

Let  $\mathbf{L} := \{\mathcal{L}_0, \mathcal{L}_1, \dots\}$  be a set of different loss functions (and/or regularization terms, constraint-penalties, etc.). We can compose the total loss - alias *objective loss* - for the optimization process respectively by a (weighted) sum of all individual losses, typically done in the form of:

$$\mathcal{L}_{total} := \sum_{i=0}^{|\mathbf{L}|-1} \alpha_i \mathcal{L}_i \quad (7.31)$$

However, the performance of the model is absolutely sensitive to the choice of the weighting values  $\alpha$ ; obviously in addition to the individual loss values themselves. In fact, each loss function can yield different loss magnitudes as they can operate on different scales, complexities, etc. Hence, even small loss magnitudes can hold significant value, while large magnitudes do not necessarily indicate greater importance. Therefore, the significance of a loss cannot (and should not) be solely determined by its magnitude.

There exist several methods for composing weighting schemes, such as using multi-objective optimization (Sener and Koltun, 2018), gradient normalization (Chen et al., 2018), or using uncertainty (Kendall, Gal, and Cipolla, 2018). Yet, these methods are not always suited for a single-task setting, which is why multi-loss weighting with coefficient of variations (CoV-Weighting) was proposed (Groenendijk et al., 2021). CoV-Weighting is based on the idea that a loss with a constant value should not be optimized further, but variance alone is insufficient as a metric since losses with larger scales can also have larger variances. Although in the following we will summarize the math behind CoV-Weighting, we highly recommend reading the original paper for a more in-depth understanding and motivation. Overall, we will be applying CoV-Weighting to our multi-loss objective (consisting of  $\mathbf{L}$ ).

Let  $\mu, \sigma$  respectively denote the mean and standard variation. First, deriving observations from a ratio scale is required for a valid calculation of the coefficient of variation, and thus the authors propose scaling the losses for each time-step  $t$  via:

$$\ell_{i,t} := \frac{\mathcal{L}_{i,t}}{\mu_{\mathcal{L}_{i,t-1}}} \quad (7.32)$$

The coefficient of variation  $c_{\ell_i}$  for a given loss then divides the standard deviation of  $\ell_i$  by its mean value:

$$c_{\ell_i} := \frac{\sigma_{\ell_i}}{\mu_{\ell_i}} \quad (7.33)$$

The corresponding weight  $\alpha_i$  at time-step  $t$  is then:

$$\alpha_{i,t} := \frac{c_{\ell_{i,t}}}{\sum_j c_{\ell_{j,t}}} = \frac{\sigma_{\ell_i}}{\mu_{\ell_i} \sum_j c_{\ell_{j,t}}} \quad (7.34)$$

For computational efficiency, mean and standard deviations are calculated incrementally without the need of having to store past observations using Welford's algorithm (Welford, 1962). They first keep track of the exponential moving averages:

$$\begin{aligned} \mu_{\mathcal{L},t} &= (1 - \frac{1}{t})\mu_{\mathcal{L},t-1} + \frac{1}{t}\mathcal{L}_t \\ \mu_{l,t} &= (1 - \frac{1}{t})\mu_{l,t-1} + \frac{1}{t}l_t \end{aligned} \quad (7.35)$$

and then calculate the incremental standard deviation  $\sigma_t := \sqrt{\mathbf{M}_t}$ , via:

$$\mathbf{M}_t = (1 - \frac{1}{t})\mathbf{M}_{t-1} + \frac{1}{t}(l_t - \mu_t)(l_t - \mu_{t-1}) \quad (7.36)$$

Unlike the paper, however, we will use a slightly different but mathematically equivalent formula to determine the standard deviation, yielding a somewhat more convenient version for code (Finch, 2009; Weisstein, n.d.):

$$\mathbf{M}_t = (1 - \frac{1}{t})(\mathbf{M}_{t-1} + \frac{1}{t}(l_t - \mu_{l,t-1})^2) \quad (7.37)$$

We can further limit the exponential averaging window size  $t$  to  $t_{max}$  by simply setting  $t \leftarrow \max(t, t_{max})$ , to eliminate older scores. Furthermore, we recommend having a ramp-up phase for the first  $t_{min}$  epochs to allow for more stable weight scores, especially as scores at the start of the training may change dramatically throughout the first epochs.

However, there are two minor possible concerns with CoV-Weighting: (1) it assumes that all losses are *equally* essential to optimize. The loss weights are determined based on the properties seen during model training, incorporating a measure of uncertainty to balance the losses, but the importance of losses is viewed in terms of model optimization rather than the user's preferences (e.g. priorities). For example, perhaps the classification- and reconstruction loss should be given higher importance and weight than a secondary loss like feature decorrelation or -contrast. (2) It is based on the assumption that a loss term is satisfied when its variance approaches zero. It does not consider, however, that not optimizing it anymore may again worsen that loss and "unlearn" what it had already learned. Furthermore, losses may not always be independent of one another; for example, only via co-occurring optimization may two specific losses result in the most optimal model performance.

As a result, we recommend the CoV-Weighting to be slightly rewritten by normalizing the weights to a mean of one and accommodating loss priorities using a combination of priority- and CoV-Weighting scores:

$$\begin{aligned}
 \hat{\alpha}_{i,t} &:= p_i(1 + c_{\ell_{j,t}} - \mathbb{E}[\alpha_{i,t}]) \\
 &= p_i\left(1 + c_{\ell_{i,t}} - \frac{1}{|\mathbf{L}|} \sum_j \frac{c_{\ell_{j,t}}}{\sum_k c_{\ell_{k,t}}}\right) \\
 &= p_i\left(1 + c_{\ell_{i,t}} - \frac{1}{|\mathbf{L}| \sum_j c_{\ell_{j,t}}} \sum_j c_{\ell_{j,t}}\right) \\
 &= p_i\left(1 + c_{\ell_{i,t}} - \frac{1}{|\mathbf{L}|}\right)
 \end{aligned} \tag{7.38}$$

where  $p_i > 0$  is a priority weight. The new weight is therefore proportional to the average of user priority and model loss importance. Now, the model can be optimized to not only minimize losses but also achieve specific objectives that are important to the user / to a use case. This simple modification provides us with great flexibility and adaptability during the training process. Note, that an intuitive alternative approach would be to simply multiply each loss by  $|\mathbf{L}|$  (see Chapter 9.2).

### Take-Away

When solving for various objectives, multiple loss functions are typically employed. In general, not all losses are equally difficult, relevant or important to optimize during the training. Here, a key factor that determines the overall effectiveness is the selection of appropriate weights for these individual losses. By incorporating proper adaptive & automated weighting, we can enhance training efficiency and improve model performance.

### Keywords

Multiple Loss Functions • Optimizing Objectives • Proper Weights • Automated Weighting • Efficient Training

## 7.5.2 Neural Network Forward Call

Let  $k, m, n \in \mathbb{N}_+$ . To utilize DNNs for generating embeddings, we need to modify the network's forward call logic to handle matrix inputs. In our case, the input is a matrix  $X$  (or a batch of matrices), represented as  $X \in \mathbb{R}^{k \times m}$ . Traditional neural networks are designed to take input as vectors or batches of vectors. Thus, it is not possible to just "feed" batches of matrices into the model. Hence, we redefine the forward pass function of the DNN for a given layer as the matrix product of  $U$  and the layer's weight matrix  $W \in \mathbb{R}^{m \times n}$ , followed by adding (via broadcasting) a bias vector  $b \in \mathbb{R}^n$ . This is then passed through a non-linear activation function  $g$ :

$$h(X) := g(XW + \vec{b}) \tag{7.39}$$

The very first layer will have  $X = U$  (the child-matrix) as input, where  $k = |\Theta_\omega|$  corresponds to the number of total child nodes from a given node  $\omega$ , and, respectively,  $m = \dim(\vec{v}_{\theta,feat} \oplus \vec{v}_{\theta,child})$  equals the respective encoding size. In particular, notice how the weight dimensions are independent of the number of child nodes,

and solely depend on the child-encoding size/dimension. Note that when dealing with batches, Equation 7.39 is simply applied independently to each matrix along the batch dimension.

### 7.5.3 Embedding Layer

Let  $h_k^*(\mathbf{U}_\omega) = h_k(h_{k-1}(\dots(h_0(\mathbf{U}_\omega))))$  denote the hidden state of forwarding  $\mathbf{U}_\omega$  through multiple layers up to the embedding layer  $k$ . We define  $z_{\mathbf{U}_\omega}$  as the one-dimensional vector sum along the column-dimension with components:

$$z_{\mathbf{U}_\omega}^{[i]} := \sum_{j=0}^{|\Theta_\omega|-1} h_k^*(\mathbf{U}_\omega)^{[j,i]} \quad (7.40)$$

We explicitly use the *sum* and not the average during pooling, since the mean does average out information about the number of child nodes.

Further, let the weighted and biased standardized vector be denoted by  $\pi(\vec{z}_{\mathbf{U}_\omega})$ :

$$\pi(\vec{z}_{\mathbf{U}_\omega}) := \vec{w} \circ \left( \frac{\vec{z}_{\mathbf{U}_\omega} - \mu_{\vec{z}_{\mathbf{U}_\omega}}}{\sigma_{\vec{z}_{\mathbf{U}_\omega}}} \right) + \vec{b} \quad (7.41)$$

with  $\vec{w}, \vec{b}$  respectively trainable weight and bias terms;  $\circ$  is the Hadamard product. This is based on the concept of layer normalization with element-wise affine transformations. Then, based on an activation function  $g$ , we define the embedding function  $f_{emb}(\cdot)$  as:

$$f_{emb}(\mathbf{U}_\omega) := g(\pi(\vec{z}_{\mathbf{U}_\omega})) \quad (7.42)$$

Note, how if  $g$  can also be chosen as being the identity function. Overall, this approach, however, presupposes that all sub-nodes are equally important, which may not be the case. Some sub-nodes (alone or in combination with others) are likely to be more relevant and decisive, and should thus contribute to and influence the embedding more strongly. A solution often applied in graph neural networks is the use of the *attention* mechanism (Veličković et al., 2017), that assigns attention scores (i.e. importance weights)  $\alpha$  to each neighboring graph. We can incorporate this idea, and obtain embeddings using a weighted attention-based weighted aggregation:

$$a_{\mathbf{U}_\omega}^{[i]} := \sum_{j=0}^{|\Theta_\omega|-1} \alpha^{[j]} h_k^*(\mathbf{U}_\omega)^{[j,i]} \quad (7.43)$$

and finally use:  $f_{emb}(\mathbf{U}_\omega) \triangleq g(\pi(\vec{a}_{\mathbf{U}_\omega}))$ . For simplicity, we will use the notation  $\mathbf{z}_\omega := f_{emb}(\mathbf{U}_\omega)$  to denote the embedding (i.e. the representation vector) of  $\omega$ . Also, recall that  $f_{emb}(\mathbf{U}_\omega) = f_{emb}(U_\omega)$  for any matrix representation  $U_\omega$  of  $\mathbf{U}_\omega$ .

### 7.5.4 Extraction Layer

Given a one-dimensional embedding vector  $\mathbf{z}_\omega := f_{emb}(U_\omega)$ , we wish to build a matrix with the same dimensions as the input matrix  $U_\omega \in \mathbb{R}^{|\Theta_\omega| \times m}$ . However, when we use the pooling operation (i.e. the mean operation), we lose information on the original dimensionality value of  $|\Theta_\omega|$  (e.g. the number of child nodes). The reconstruction step, on the other hand, is essential for the implementation of an auto-encoder and calculating the reconstruction loss. Furthermore, we must preserve the row identity relationship to each child, which could easily be overlooked during the

pooling procedure. To conclude, we would be attempting to build a matrix  $\hat{U}_\omega$  from a vector  $\mathbf{z}_\omega$  derived from the previous input matrix  $U_\omega$ . This, however, results in information loss. As a result, we propose doing the reconstruction using  $h_k^*(U_\omega)$  (which determines  $\mathbf{z}_\omega$ ) instead. The idea is illustrated in Figure 7.12 below:

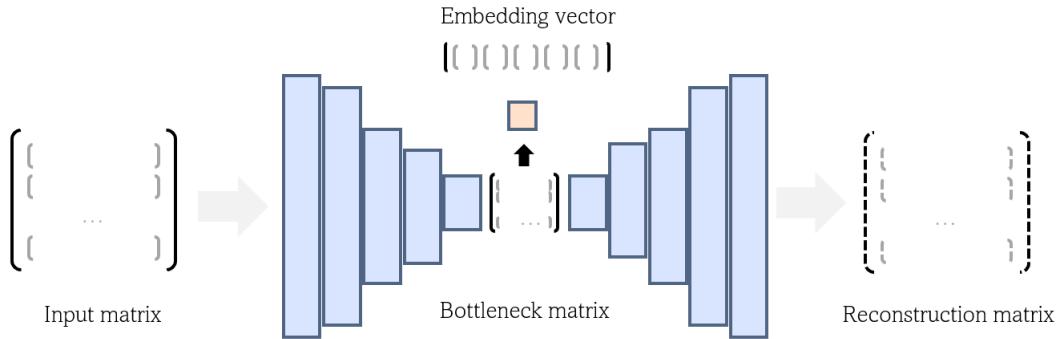


FIGURE 7.12: Bottleneck reconstruction: We use the last latent bottleneck matrix to rebuild the original input rather than the latent representation vector. This prevents a strong information loss during the reconstruction process. The reconstruction is  $U_\omega \rightarrow h_k^*(U_\omega) \rightarrow \hat{U}_\omega$ .

### 7.5.5 Batching

For batching, we simply concatenate the respective child-matrices  $U_{\omega 0}, U_{\omega 1}, \dots$  along a new third dimension and (dynamically) pad them equivalently to equal size, which we will refer to as  $U^*$ . However, special care must be taken when computing averages (or attention scores) over padded sequences, as we do not wish to perform calculations on (potentially variable-sized) padding entries.

### 7.5.6 Model Architecture

Instead of using a single (huge) AE, we will build upon the concept of *divide-and-conquer* by recursively defining and interconnecting multiple smaller AEs (one per tree level). That is, the embedding output of a tree's level-associated AE will be (part of) the input of the AE for the respective level above (Figures 7.13 & 7.14). Information is, thus, propagated upwards; using a bottom-up approach. That is, for each child of a given node, first the corresponding embedding is calculated and concatenated with the corresponding child's feature vector, yielding the child-matrix  $U$ . Therefore, it is obvious that the representation of parent nodes - especially that of the root node - can only be calculated once the embeddings of all child nodes have been calculated. In other words, the algorithm begins by computing the embeddings of the lowest levels and then propagating the results upwards, with higher-level representations encompassing those below them. And so, utilizing AEs at each level of the tree aids in mitigating information loss or corruption that may occur per level. This architecture is defined recursively.

Therefore, by repeatedly applying this recursive model architecture to the whole input tree, we obtain a complete DNN embedding model (Figure 7.14). The resulting final embedding of the tree is simply the output of the highest level (i.e. root node). Such a recursive approach also enables us to assign different weights to the losses (and also different losses) for each hierarchy level, providing so more adaptability;

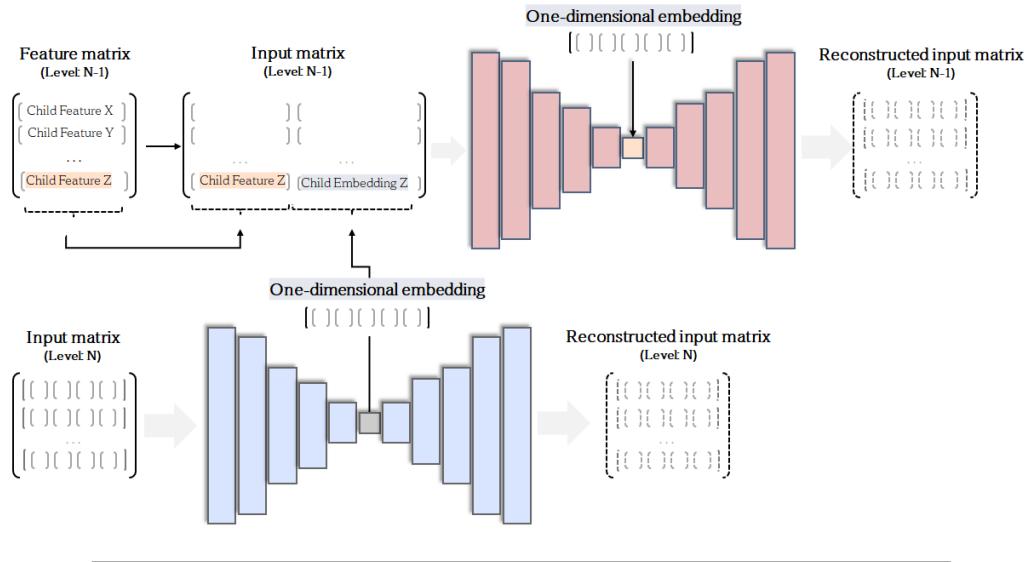


FIGURE 7.13: The recursive model structure. The output of the lower model (blue) is part of the input for its direct superior model (red).

e.g. giving more emphasis/importance to the losses on the upper levels and less focus/importance on the lower ones.

Let  $\mathcal{L}_{total}^{[j]}$  be the loss as defined in Equation 7.31 for a given level  $j$ . Then we define the total model's loss as:

$$\mathcal{L}_{total}^{model} := \sum_{j=0}^{\#levels-1} w_j \mathcal{L}_{total}^{[j]} \quad (7.44)$$

where  $w_j$  is an (optional) associated weight for level  $j$ . In other words, the total loss of the model consists of the weighted sum of all level losses.

Alternatively, we can also an even more flexible variation using weights  $\alpha_{j,i}$ :

$$\mathcal{L}'_{total}^{model} := \sum_{j=0}^{\#levels-1} w_j \sum_{i=0}^{|L|-1} \alpha_{j,i} \mathcal{L}_i \quad (7.45)$$

### 7.5.7 Example

Figure 7.15 shows exemplary an illustration of how the input matrix and the node embedding are retrieved for a given layer on a node with three child nodes.

First, the child node features and -encodings of all child nodes are concatenated into a single matrix and then passed through the DNN (i.e. AE). The respective embedding layer is then used as the child encoding of the child nodes' parent node. The algorithm then subsequently continues to propagate information upwards for all remaining nodes in the same manner until it reaches the root node.

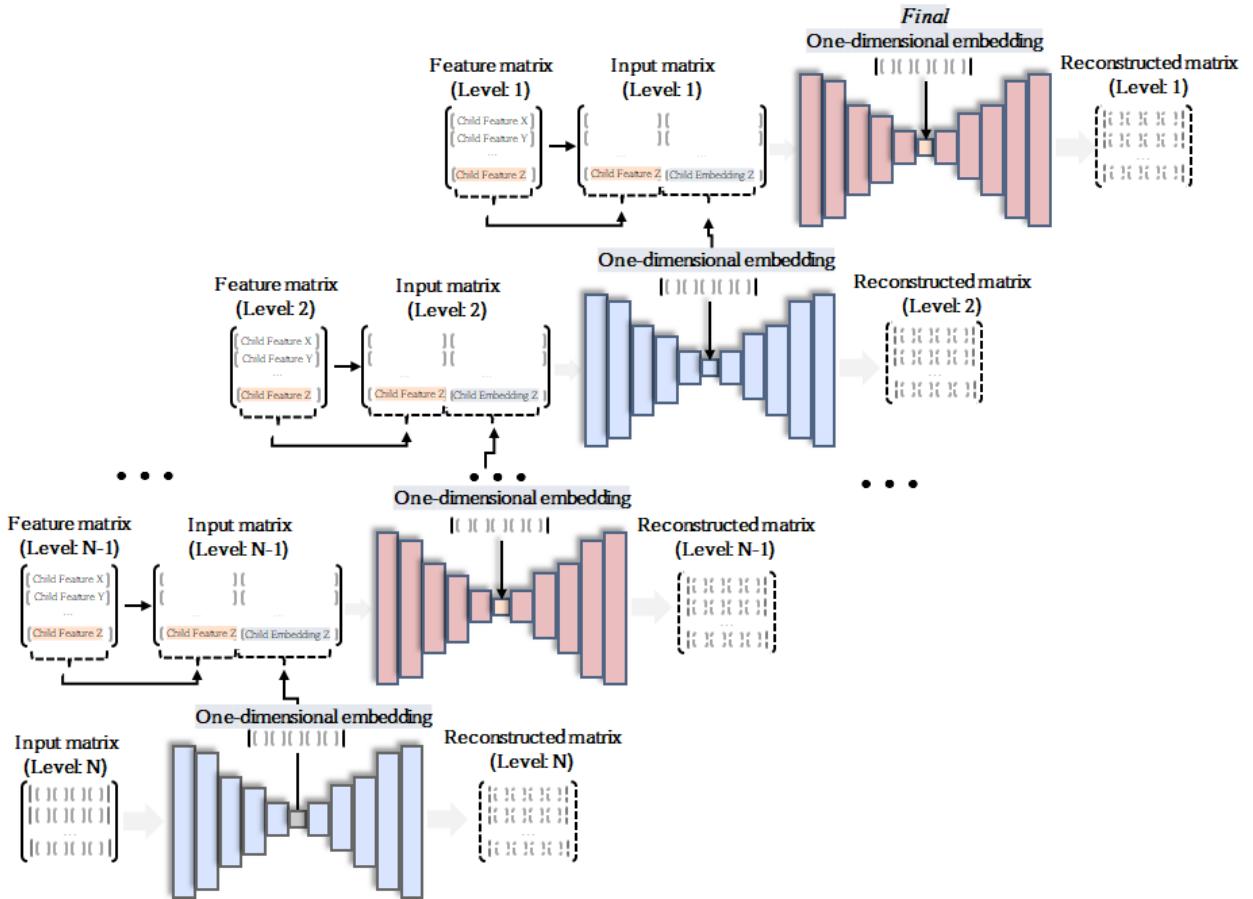


FIGURE 7.14: The recursive composition of the HDB model structure.

### Take-Away

Instead of trying to solve the problem at once, we adopt a "divide and conquer" strategy by arranging multiple models in a recursive/hierarchical manner. This gives us more control over the learning process and helps us uncover and determine more effective intermediate vector representations. These representations are passed on to higher levels until reaching the root node, ultimately representing the entire tree structure.

### Keywords

Divide and Conquer • Hierarchical Stacking • Recursive Models

### 7.5.8 Model Improvements

With a clear understanding of the basic architecture and model, we can now concentrate on implementing effective adaptations that yield even greater improvements. These adaptations may complicate the model definition and add to the computational load, but the improved model quality justifies their use.

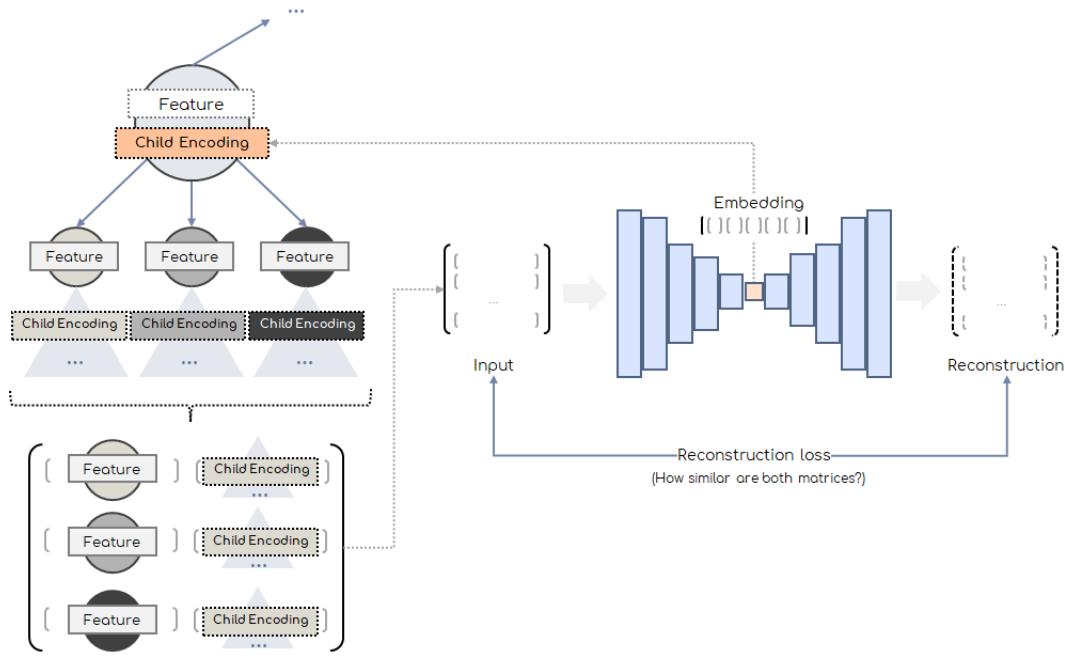


FIGURE 7.15: Exemplary illustration of input matrix generation based on child nodes for a given layer.

### Adaptive Look-up Embeddings

Instead of using a fixed encoding scheme - such as *One-Hot-Encoding* - we can integrate an adaptive look-up embedding system (similar to embedding-layers as in Kocmi and Bojar, 2017 and Neishi et al., 2017).

The input to our model is, therefore, no longer "simply" the concatenation of the node-encoding plus child-embedding (Equation 7.5), but the concatenation of an *adaptive* embedding from a look-up table. These embeddings are "adaptive" in the sense that they support gradient calculation and, thus, can be optimized e.g. by the back-propagation pass. Initially, these vector embeddings are randomly initialized and tend to be pairwise orthogonal since the dot product of two randomly sampled vectors in high dimensions converges to zero with increasing size. Each node (i.e. token) then holds a unique index to access its corresponding vector through a look-up table. Although the look-up itself is not differentiable, the vector values are and since we have a bijective function from tokens to embedding vectors we can so optimize the embedding corresponding to each node (Figure 7.22).

It may also make sense to *freeze* the embedding layers (e.g. after a portion of the total training epochs) to allow the model to focus only on feature values and sub-embedding representations to prevent it from merely optimizing categorical embeddings without consideration of the other features.

In addition, the size of the learnable categorical embeddings can be determined by the number of unique categories. Specifically, if a category contains a large number of entities, it may require a higher dimensional space to effectively capture and encode all semantic relationships and differences. Conversely, if a category contains only a few entities, it may require fewer dimensions. A commonly used convention/rule-of-thumb for a good starting point is to use approximately the square root of the number of distinct categories. Thus, to determine the embedding size  $n_c$  for a category  $c$  with  $k$  different unique entities and a fixed maximum

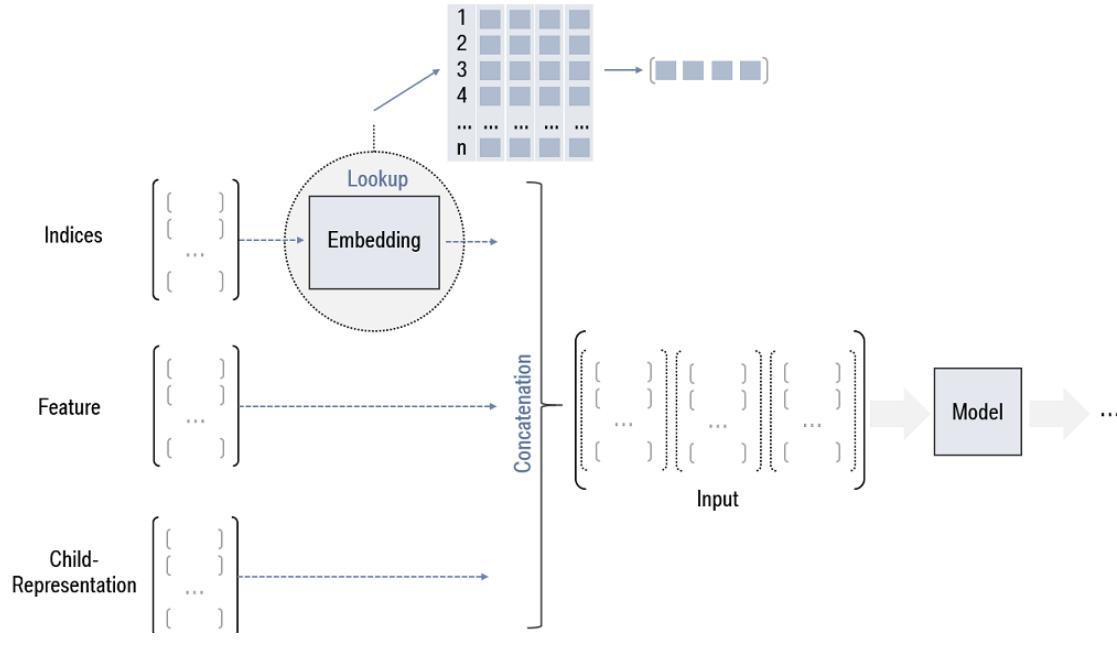


FIGURE 7.16: Integrating adaptive look-up embeddings.

dimension  $d$ , we suggest using one of the following two bounded formulas:

$$n_c = \min(k, d), \quad (7.46)$$

$$n_c = \min(\sqrt{k}, d). \quad (7.47)$$

We will be using Equation 7.46; it ensures that the embedding size does not exceed the maximum dimension  $d$ , regardless of the number of entities in the category. Overall, both formulas allow for flexibility in the embedding size based on the number of entities in each category.

### Take-Away

Adaptive look-ups allow the model to dynamically determine and optimize embeddings. Unlike one-hot-encodings, these representations are dense, can encode semantic information, and are of much lower dimension.

### Keywords

Adaptive Look-Up Embeddings • Dense Representations • Lower Dimension

## Residual Attention Mechanism

Drawing inspiration from transformers and attention mechanisms (Vaswani et al., 2017), we can enhance our model by incorporating *self-attention*, specifically using the *scaled dot-product*. This approach enables us to prioritize important parts of the input by calculating attention scores and scaling the corresponding features according to their relative significance/importance. To address the issue of information loss we can introduce residual network connections (He et al., 2016; Zagoruyko and

Komodakis, 2016) (as also utilized and adopted in transformers). These direct connections allow for bypassing certain layers and can help prevent vanishing gradients, thus increasing training stability. Residual connections also serve as a means of preserving the original input state representation. That is, first, we apply self-attention to the input vector using the scaled dot-product. Then, we add the input vector to the attention vector and perform layer-normalization. Finally, we pass the result through a feed-forward pass and add the previously normalized scores to the output vector of the forward pass (Figure 7.17). The result is then passed through an activation function (e.g. to the next layer).

Residual connections, in their most basic form, are defined by  $y := F(x) + x$ , and so the dimension of  $F(x)$  must be that of  $x$ . Equal dimension can be achieved e.g. by either maintaining the same dimension throughout all layers of the model or utilizing residual blocks where equal dimensions are enforced; residual connections are then only applied within each block and not between different blocks.

We further also incorporate *layer-normalization* in all layers, except the last one, of the decoder. Skipping the last one is essential to ensure that the output range of the decoder's last layer is the same as that of the input vectors. Otherwise, the decoder may not be able to accurately reconstruct the original input vector. Although one could argue that layer-normalizers usually use trainable scaling and bias factors, we still decided not to normalize the last layer; as this does still not fully mitigate or circumvent the problem.

### Take-Away

To enable the model to effectively utilize and identify the most relevant features, we can implement a residual self-attention mechanism.

### Keywords

Self-Attention Mechanism • Relevance Identification • Feature Utilization

## Inception Networks

*Inception networks* were initially proposed by Google and have seen multiple improvements and variations since (Szegedy et al., 2015; Szegedy et al., 2016; Szegedy et al., 2017). The motivation was to increase effectiveness, and performance and to leverage the computational budget of training deep neural networks, in particular of CNN networks and computer vision, but inspiration of it has also been found in other areas such as the *Dueling Deep Q-network* (DDQN) in reinforcement learning (Wang et al., 2016c).

The main idea is to run parallel network blocks that share the same input and are concatenated into one output, instead of running one huge layer block. We can build upon the idea of inception networks and DDQN and use the same principles to better learn (i.e. auto-encode) the individual components - node embedding, features, child embeddings - of the input vectors.

At the end of a decoder module, we add three concurrent layers respectively and use their concatenated output as the regression target (Figure 7.18). Although it might look like a "just" simple change in the network's architecture, inception networks are indeed very powerful. Additionally, in doing so we can apply different activation and transformation functions to each of the components, which is especially useful e.g. if their value-domains differ and unlike re-scaling is needed.

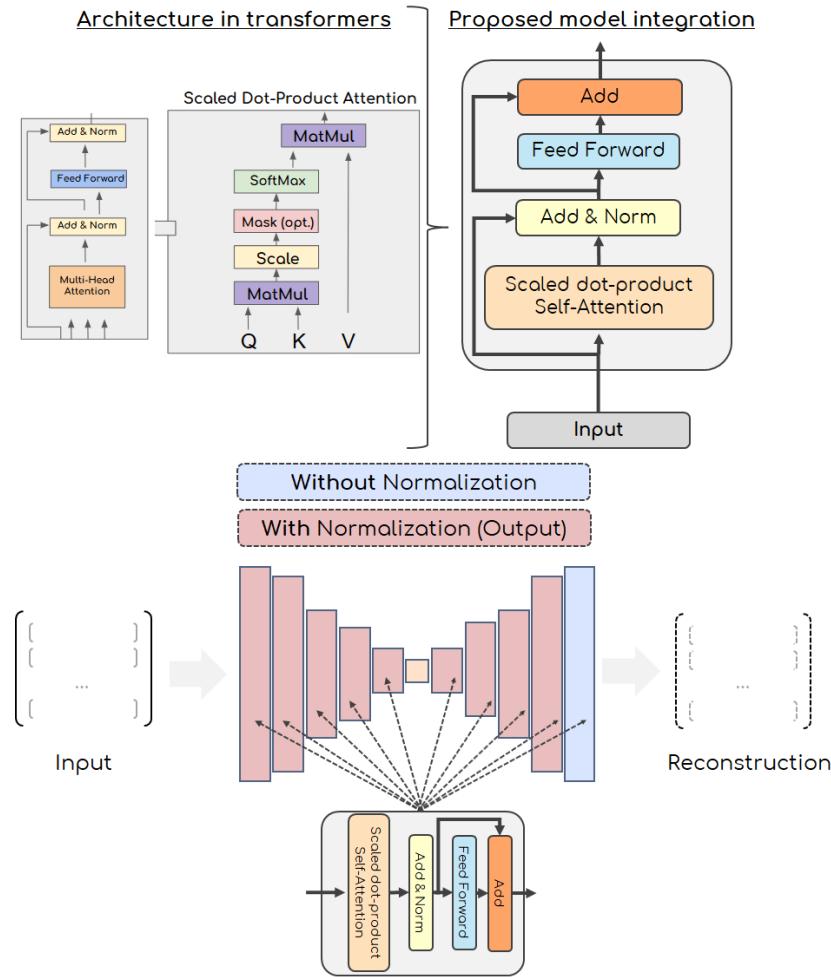


FIGURE 7.17: Integrating residual self-attention and normalization [part of the figure shows an extract from the transformer architecture (Vaswani et al., 2017)].

In fact, it is also absolutely possible to use inception networks within the encoder/decoder modules in any other combination, size, and complexity.

### Take-Away

We can enhance the decoder by allowing it to focus more on rebuilding individual parts using the inception strategy.

### Keywords

Enhanced Decoder • Inception Strategy • Focus Enhancement

## Siamese Networks

*Siamese* neural networks (SiaNNs) are neural network models that use several - typically two - parallel and identical modules with shared weights. SiaNNs aim to learn hidden representations of input vectors. The output generated by a Siamese neural network can be interpreted as the semantic similarity, e.g. between two given input

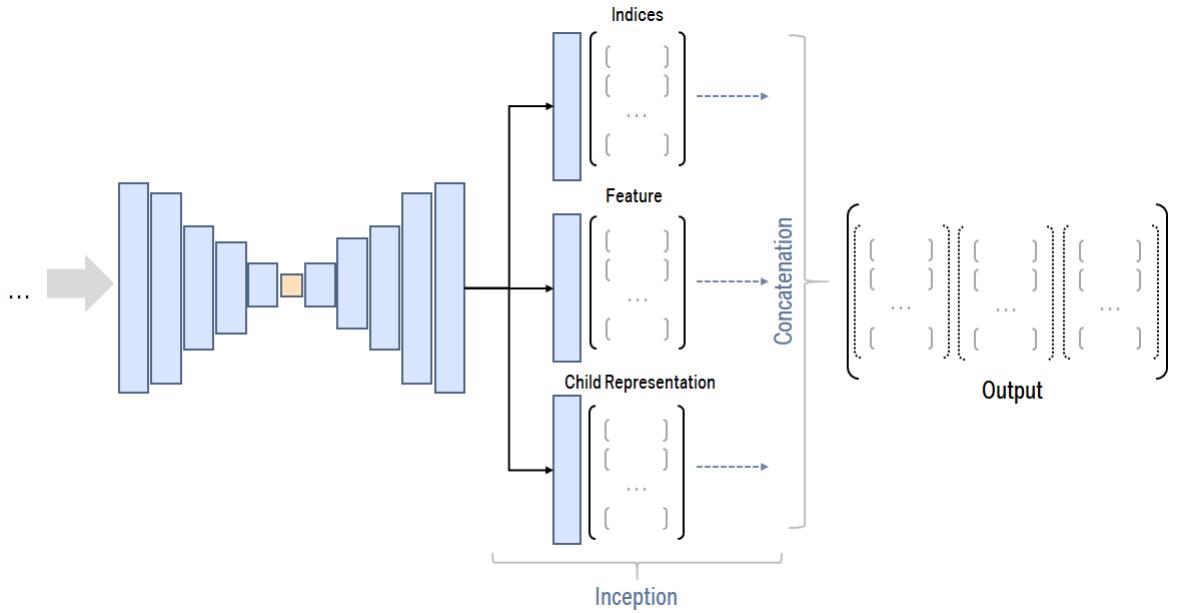


FIGURE 7.18: Applying feature-inception to the model.

vectors (Wang et al., 2016c). SiaNNs have also been used in hierarchical models on class-level trees, where neighboring classes are recursively merged to encode global context information (Ge, 2018).

We propose to incorporate and use SiaNNs in conjunction with self-attention by passing the raw input vector along with the output of its attention mechanism through a SiaNN, and then use their average as the input for the subsequent module/layer (Figure 7.19).

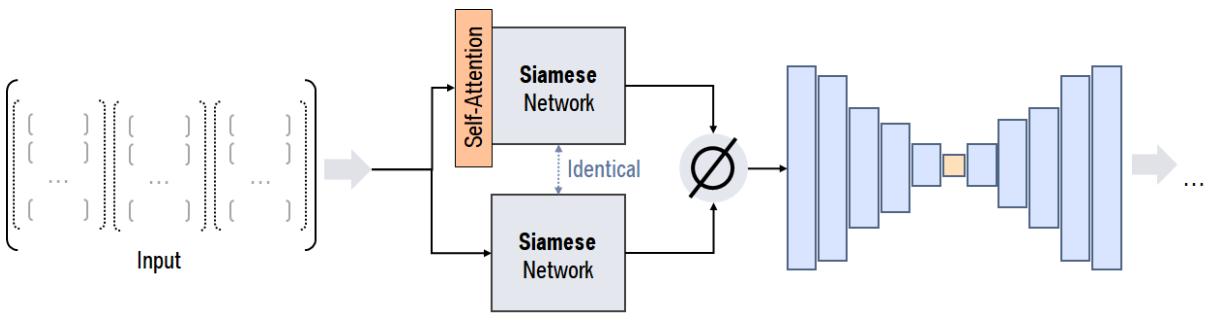


FIGURE 7.19: Applying Siamese neural networks and self-attention to the model.

Yet, SiaNNs also come with drawbacks such as increased computational and memory requirements. Nonetheless, it is worth considering that they can also serve as an efficient alternative to utilizing two completely different parallel DNNs (e.g., in large inception networks). This, in fact, even requires fewer weights since the Siamese weights are reused; unlike having two separate DNNs.

Overall, SiaNNs have been widely employed in various tasks (Chicco, 2021). In general, SiaNNs use similarity measures and pairwise comparisons to learn data representation. They compare objects using identical DNNs and calculate similarity scores. They can be leveraged and trained to maximize similarity between similar

objects and minimize it between dissimilar ones. Often, different augmentations of the same input are fed through both SiaNN components, and an equal representation is trained to be learned. Instead, we here apply self-attention as our SiaNN's input "variation" to learn a robust representation, and then take the average of both output representations to use it as the propagating output representation.

#### Take-Away

Siamese networks (1) feature fewer weights than two separate models and (2) are generally considered more robust to class imbalance and lack of data (as in One-shot learning applications), i.e., few occurrences per class are sufficient for Siamese networks to recognize them.

#### Keywords

Feature Efficiency • Robustness to Class Imbalance • One-Shot Learning • Self-Attention Siamese Input

### Mean Teacher Networks

Different data augmentations or transformations on the same input data should not influence the predictions of models and result in equal (or very similar) representations (Weng, 2021). Consistency Regularization (alias Consistency Training) is a field concerned with trying to maintain this assumption. It has been applied to self- and semi-supervised methods, particularly for utilizing unlabelled data whilst learning in a supervised manner (Ouali, Hudelot, and Tami, 2020; Weng, 2019).

Mean teachers (Tarvainen and Valpola, 2017) is an improvement over previous methods such as  $\pi$ -model (Sajjadi, Javanmardi, and Tasdizen, 2016) and Temporal Ensembling (Laine and Aila, 2016). The  $\pi$ -model uses an unsupervised loss to minimize the difference between two data passes with transformations. Instead, Temporal Ensembling maintains an exponential moving average of model predictions per epoch to improve performance. Mean Teacher Networks, on the other hand, maintain a moving average of the weights in the model (which is not necessarily limited to being averaged *per epoch*).

We can complement (or alternatively even replace) the SiaNNs respectively by a Mean Teacher architecture that uses a teacher and student network. Alternatively, using Dual Students<sup>5</sup> - a modification of Mean Teachers that replaces the teacher with a second student (Ke et al., 2019), is also an option.

#### Take-Away

Mean Teacher Networks may be used to better ensure that randomness inside a model does not change the network's outputs and latent representations (for the same input).

#### Keywords

Mean Teacher Networks • Consistent Outputs • Randomness Control

<sup>5</sup>Dual Students is an interesting/promising approach also particularly for semi-supervised settings where only part of the data is labeled.

## Complex-Valued Neural Networks

Instead of simply using conventional real-valued-based NNs, we also integrate the proposed approach of *complex-valued* networks. Complex-valued networks may have higher computational overhead but potentially offer better neural expression. To optimize the performance, we can adjust the number of layers in the DNNs as needed. The (angular) weights are initialized randomly within the range of  $[-\pi, \pi]$  to cover the entire angles of the unit circle. Note that, although a range of  $[0, 2\pi]$  is mathematically equivalent, the difference is that in the former case, the mean value is centered around zero, and in the latter case not. This subtle difference can make a difference in the case of DL as it has an impact on the back-propagation gradients; also, symmetry around zero is always a good convention.

### Take-Away

We constrain the complex-valued network weights at  $[-\pi, \pi]$  by applying a modulo operation. The complex-valued network can be directly created by simply replacing the real-valued neurons.

### Keywords

Complex-Valued Neural Network • Weight Constraints • Modulo Operation

## Label Smoothing

Label smoothing (Szegedy et al., 2016) is a well-known technique that introduces noise from a distribution to labels, typically to prevent over-fitting by not allowing the model to become over-confident. In essence, it introduces a (small) degree of uncertainty by attributing non-zero probabilities to incorrect classes during training, thus preventing the model from overly relying on its training data and, hence, mitigating the risk of overfitting. Recent work has shown that label smoothing encourages instances from the *same* class to group in tight clusters, thereby resulting in loss of information about resemblances between instances of *different* classes (Müller, Kornblith, and Hinton, 2019); and is so *not* good for knowledge distillation. Lukasik et al., 2020, however, argued that this observation applies specifically to non-noisy labels. Instead, they showed that label smoothing competes with loss-correction under label noise; and when distilling models from **noisy data**, incorporating label smoothing for the teacher proves advantageous.

We can, thus, use label smoothing to explore the trade-off between identifying tighter, more compact clusters versus less tight, more overlapping clusters; and it also helps us in addressing label uncertainty and noise. Given a noise distribution  $u(y|x)$  over classes  $y \in Y$  and data  $x \in X$  (e.g a uniform distribution), the smoothed label loss for a prediction  $q(y|x)$  and a real ground truth  $p(y|x)$  is:

$$\begin{aligned}
 \mathcal{L}' &= - \sum_{x_i \in X} \sum_{y_j \in Y} \left[ (1 - \epsilon) p(y_j|x_i) + \epsilon u(y_j|x_i) \right] \log q(y_i|x_i) \\
 &= - \sum_{x_i \in X} \left\{ \left[ (1 - \epsilon) \sum_{y_j \in Y} p(y_j|x_i) \log q(y_i|x_i) \right] + \left[ \epsilon \sum_{y_j \in Y} u(y_j|x_i) \log q(y_i|x_i) \right] \right\} \\
 &= - \sum_{x_i \in X} \left\{ (1 - \epsilon) \mathcal{H}_i(p, q) + \epsilon \mathcal{H}_i(u, q) \right\}
 \end{aligned} \tag{7.48}$$

where  $\epsilon \in [0, 1]$  denotes a smoothing factor, and  $\mathcal{H}(., q)$  the distribution entropy of  $q$  given a target distribution:

$$\mathcal{H}_i(g, q) := \sum_{y_j \in Y} p(y_j|x_i) \log q(y_i|x_i) \tag{7.49}$$

In particular, if  $q(y_j|x_i)$  is the Softmax function on the logit  $\hat{q}(y_j|x_i)$ , we can further simplify  $\mathcal{H}(g, q)$ , for  $g \in \{p, u\}$ , to:

$$\begin{aligned}
 \mathcal{H}_i(g, q) &= \sum_{y_j \in Y} g(y_j|x_i) \log \left( \frac{e^{\hat{q}(y_j|x_i)}}{\sum_{y_k \in Y} e^{\hat{q}(y_k|x_i)}} \right) \\
 &= \sum_{y_j \in Y} g(y_j|x_i) \left[ \hat{q}(y_j|x_i) - \log \left( \sum_{y_k \in Y} e^{\hat{q}(y_k|x_i)} \right) \right]
 \end{aligned} \tag{7.50}$$

If  $\mathbf{y}$  is an instance's only given class label, such that  $p(\mathbf{y}|x) \in \{0, 1\}$  is an *indicator* function, with  $\forall x_i \in X \exists! \mathbf{y} \in Y : p(\mathbf{y}|x_i) = 1 \wedge \forall y_j \neq \mathbf{y} : p(y_j|x_i) = 0$ , then the above formula reduces to:

$$\mathcal{H}_i(g, q) \triangleq \hat{q}(\mathbf{y}|x_i) - \log \left( \sum_{y_k \in Y} e^{\hat{q}(y_k|x_i)} \right) \tag{7.51}$$

Likewise, for a uniform distribution  $u = \frac{1}{|Y|}$  it follows:

$$\begin{aligned}
 \mathcal{H}_i(u, q) &= \sum_{y_j \in Y} \frac{1}{|Y|} \left[ \hat{q}(y_j|x_i) - \log \left( \sum_{y_k \in Y} e^{\hat{q}(y_k|x_i)} \right) \right] \\
 &= \mathbb{E}_{y_j \in Y} \left[ \hat{q}(y_j|x_i) - \log \left( \sum_{y_k \in Y} e^{\hat{q}(y_k|x_i)} \right) \right]
 \end{aligned} \tag{7.52}$$

And so, for both, uniform noise and one-hot-encoded labels, we have:

$$\begin{aligned}
& (1 - \epsilon)\mathcal{H}_i(p, q) + \epsilon\mathcal{H}_i(u, q) \\
&= (1 - \epsilon) \left[ \hat{q}(\mathbf{y}|x_i) - \log \left( \sum_{y_k \in Y} e^{\hat{q}(y_k|x_i)} \right) \right] + \epsilon \mathbb{E}_{y_j \in Y} \left[ \hat{q}(y_j|x_i) - \log \left( \sum_{y_k \in Y} e^{\hat{q}(y_k|x_i)} \right) \right] \\
&= \mathbb{E}_{y_j \in Y} \left[ (1 - \epsilon) \left( \hat{q}(\mathbf{y}|x_i) - \log \left( \sum_{y_k \in Y} e^{\hat{q}(y_k|x_i)} \right) \right) + \epsilon \left( \hat{q}(y_j|x_i) - \log \left( \sum_{y_k \in Y} e^{\hat{q}(y_k|x_i)} \right) \right) \right] \\
&= (1 - \epsilon)\hat{q}(\mathbf{y}|x_i) + \epsilon \mathbb{E}_{y_j \in Y} [\hat{q}(y_j|x_i)] - \log \left( \sum_{y_k \in Y} e^{\hat{q}(y_k|x_i)} \right)
\end{aligned} \tag{7.53}$$

which gives us the loss:

$$\begin{aligned}
\mathcal{L}' &= - \sum_{x_i \in X} \left\{ (1 - \epsilon)\mathcal{H}_i(p, q) + \epsilon\mathcal{H}_i(u, q) \right\} \\
&= - \sum_{x_i \in X} \left\{ (1 - \epsilon)\hat{q}(\mathbf{y}|x_i) + \epsilon \mathbb{E}_{y_j \in Y} [\hat{q}(y_j|x_i)] - \log \left( \sum_{y_k \in Y} e^{\hat{q}(y_k|x_i)} \right) \right\}
\end{aligned} \tag{7.54}$$

and, thus, the average (i.e. mean) loss  $\bar{\mathcal{L}'}$  is:

$$\bar{\mathcal{L}'} = \mathbb{E}_{x_i \in X} \left[ \log \left( \sum_{y_k \in Y} e^{\hat{q}(y_k|x_i)} \right) - (1 - \epsilon) \underbrace{\hat{q}(\mathbf{y}|x_i)}_{\text{Prob. class label}} - \epsilon \underbrace{\mathbb{E}_{y_j \in Y} [\hat{q}(y_j|x_i)]}_{\text{Mean prob. all classes}} \right] \tag{7.55}$$

### Take-Away

Label Smoothing is a noise-introducing regularization approach for labels, where a small amount of probability mass is distributed to other classes. This can reduce overfitting and enhance the generalization of the model. Utilizing label smoothing can prevent the network from becoming overly confident.

### Keywords

Label Smoothing • Regularization • Preventing Model Overconfidence

### Focal Smoothing

Given  $\mathbf{y}$  as the ground-truth class label; let  $q^*(.,.)$  be defined as:

$$q^*(y_j|x_i) = \begin{cases} q(y_j|x_i), & \text{if } y_j = \mathbf{y} \\ 1 - q(y_j|x_i), & \text{otherwise} \end{cases} \tag{7.56}$$

where  $q(y_j|x_i)$  is the predicted probability of  $y_j \in Y$  for  $x_i \in X$ .

Focal smoothing (Lin et al., 2017) is a way of dealing with imbalanced (labeled) data. It builds upon the cross-entropy loss and for the binary case it is defined as:

$$FL^*(y_j|x_i) := - [1 - q^*(y_j|x_i)]^\gamma \log q^*(y_j|x_i) \tag{7.57}$$

where  $\gamma > 0$  is the tunable *focusing parameter* which down-weights easy examples and allows to focus more on training on hard negatives. In the binary case,  $q^*(y_j|x_i)$  is the probability of the *true* class. For multi-class, this can easily be adapted to:

$$FL(y_j|x_i) := -[1 - q(y_j|x_i)]^\gamma \log q(y_j|x_i) \quad (7.58)$$

### Combining Focal- & Label smoothing

We can combine focal- with label smoothing by incorporating  $FL(y_j|x_i)$  into  $\mathcal{H}_i(g, q)$  (Equation 7.49); and instead use:

$$\mathcal{H}_i^{[FL]}(g, q) := \sum_{y_j \in Y} p(y_j|x_i) [1 - q(y_j|x_i)]^\gamma \log q(y_j|x_i) = - \sum_{y_j \in Y} p(y_j|x_i) FL(y_j|x_i) \quad (7.59)$$

If we are using an **indicator function** (i.e. one-hot-encoded probabilities), this further reduces to applying the focal loss to the *true* label probability; thus, combining focal loss with label smoothing (Equation 7.48), this yields:

$$\begin{aligned} \mathcal{L}'' &:= - \sum_{x_i \in X} \left\{ (1 - \epsilon) \mathcal{H}_i^{[FL]}(p, q) + \epsilon \mathcal{H}_i^{[FL]}(u, q) \right\} \\ &= - \sum_{x_i \in X} \left\{ (1 - \epsilon) [1 - \hat{q}(\mathbf{y}|x_i)]^\gamma \mathcal{H}_i(p, q) + \epsilon \mathcal{H}_i^{[FL]}(u, q) \right\} \end{aligned} \quad (7.60)$$

Moreover, if using a Softmax function on the logits  $\hat{q}$ , we further have:

$$\mathcal{L}'' := - \sum_{x_i \in X} \left\{ (1 - \epsilon) [1 - \hat{q}(\mathbf{y}|x_i)]^\gamma \left[ \hat{q}(\mathbf{y}|x_i) - \log \left( \sum_{y_k \in Y} e^{\hat{q}(y_k|x_i)} \right) \right] + \epsilon \mathcal{H}_i^{[FL]}(u, q) \right\} \quad (7.61)$$

Note how we can only do the reduction trick on  $p$ , and not  $u$ , because  $u$  is not one-hot-encoded (e.g.  $u \sim \text{uniform}$ ). And so, the overall average loss for one-hot-encoded ground-truth labels and a Softmax function is:

$$\overline{\mathcal{L}''} := \mathbb{E}_{x_i \in X} \left[ (1 - \epsilon) [1 - \hat{q}(\mathbf{y}|x_i)]^\gamma \left[ \log \left( \sum_{y_k \in Y} e^{\hat{q}(y_k|x_i)} \right) - \hat{q}(\mathbf{y}|x_i) \right] - \epsilon \mathcal{H}_i^{[FL]}(u, q) \right] \quad (7.62)$$

### Take-Away

**Addressing class imbalance** can be done by modifying the traditional cross-entropy loss. This may involve reducing the loss associated with well-classified instances and increasing otherwise; i.e. distinguishing between easy and hard examples. A suitable approach in this context is the application of "focal loss," which can be enhanced further by incorporating label smoothing.

### Keywords

Class Imbalance • Altered Cross-Entropy • Focal Loss • Label Smoothing

## Dropout

Dropout is a common method used to regularize DL models and counteract overfitting. It randomly "drops out" a portion of network units; setting them to zero during training. Dropout can be applied to both input vectors (or matrices) and individual neurons to prevent units from over co-adapting (Srivastava et al., 2014). We can use dropout to try to remove and/or induce noise if the goal is to learn better *generalizable* patterns, and so avoid over-fitting or learning identity functions. Dropout is typically only active during training and, hence, omitted during evaluation or deployment. There are several main variants of Dropout in the literature:

1. **Standard dropout:** The most commonly used type of dropout. During training, neurons are randomly dropped out with a fixed probability  $p$ , and the remaining activation outputs are scaled by  $1/(1 - p)$  (Srivastava et al., 2014).
2. **Alpha dropout:** Similar to standard dropout, but the dropped elements are scaled and shifted to maintain zero mean and unit standard deviation, rather than being set to zero (Klambauer et al., 2017).
3. **Drop connect:** Similar to standard dropout, but it is applied to the connections between neurons, rather than the neurons themselves (Wan et al., 2013).
4. **Gaussian dropout:** A variant that introduces Gaussian noise to the input (e.g. Kingma, Salimans, and Welling, 2015).
5. **MaxOut:** A variant which takes the maximum value across a set of linear combinations of the input variables; can approximate any continuous function to an arbitrary precision (Goodfellow et al., 2013).

### Take-Away

Dropout is a fundamental concept that randomly sets neural entries to zero (i.e. removes them). This prevents neurons in a layer from optimizing their weights simultaneously, promoting diverse convergence towards the same goal. This is crucial as an excessive correlation between neurons can otherwise significantly compromise a model's generalization ability; also resulting in redundant neurons/models that could be more efficiently compressed/leveraged (Bonnin, 2017).

### Keywords

Dropout • Weight Decorrelation • Generalization • Regularization

## Class Imbalance

We can further account for class imbalances by applying weights that are inverse to the label frequencies. If  $\mathcal{L}_{x_i}$  denotes the absolute instance loss for a specific  $x_i \in X$ , so that the total loss is  $\mathcal{L} = \sum_{x_i \in X} \mathcal{L}_{x_i}$ , and  $y_i \in Y$  is a label, then the balanced loss (inspired from King and Zeng, 2001) is:

$$\mathcal{L} := \sum_{x_i \in X} \frac{|X|}{\#y_i|Y|} \mathcal{L}_{x_i} = \frac{|X|}{|Y|} \sum_{x_i \in X} \frac{\mathcal{L}_{x_i}}{\#y_i} \quad (7.63)$$

where  $|X|, |Y|$  are respectively the total number of instances and different labels in the entire data set,  $\#y_i$  the frequency of a particular label  $y_i$ . Since we are dealing with several hierarchical levels in our case, we define different  $\#y_i^{level}$  scores for each individual level based on the total frequency of that label occurring only in that particular (tree-) hierarchy level. Hence, the class weight is  $w_y := \frac{|X|}{|Y|\#y_i^{level}}$ . Overall, other more naive trivial, yet worse, alternatives are  $1/\#y_i$  or  $1/\sqrt{\#y_i}$ . In particular, this is because one key property of Equation 7.63 is that it has a mean weight of one:

$$\begin{aligned}\mu_w &= \frac{\sum_{x_i \in X} \frac{|X|}{\#y_i|Y|}}{|X|} = \frac{\sum_{x_i \in X} \frac{1}{\#y_i}}{|Y|} \\ &= \frac{\sum_{x_i \in X} \#y_0 \frac{1}{\#y_0} \cdot \dots \cdot \#y_i \frac{1}{\#y_i}}{|Y|} = \frac{|Y| \cdot 1}{|Y|} = 1\end{aligned}\quad (7.64)$$

In contrast, e.g.  $w_y = 1/\#y_i$  would instead have a mean of  $\mu_w = |Y|/|X|$ . Also, even the intuitive normalized adaptation  $\hat{w}_y = \frac{\#y_i^{-1}}{\sum_{y_j \in Y} \#y_j^{-1}}$  has a mean of:

$$\mu_{\hat{w}} = \frac{\sum_{x_i \in X} \frac{\#y_i^{-1}}{\sum_{y_j \in Y} \#y_j^{-1}}}{|X|} = \frac{\sum_{x_i \in X} \#y_i^{-1}}{|X| \sum_{y_j \in Y} \#y_j^{-1}} = \frac{|Y|}{|X| \sum_{y_j \in Y} \#y_j^{-1}} \quad (7.65)$$

Since  $\sum_{y_j \in Y} \#y_j^{-1} \leq |Y| \implies \mu_{\hat{w}} \geq \frac{|Y|}{|X|}$ , we know that normalizing the weight scores by the inverse label frequency would here increase the mean, and so, *not* normalizing the scores actually has a lower mean. Therefore, defining a weighted total loss via such normalized scores  $\hat{w}$  is not recommended, and even worse than setting  $w_i = 1/\#y_i$  where at least  $\mu_{\hat{w}} \leq 1$  defines a supremum. Thus, it is better to use the level-wise inverse weighting  $w_y := |X|/(|Y|\#y_i^{level})$ .

Yet, while training with frequency-inverse class weights does help against the model's bias toward dominant classes to some extent, relying solely on re-weighting the loss by inverse class frequency still often results in poor performance when dealing with real-world data with (very) large class imbalance (Cui et al., 2019). The authors argue here that, as the sample size increases, the incremental value of a new data point diminishes; and so, propose a different approach to assess data overlap by associating each sample with a small nearby region rather than a single point. They introduce the concept of *sample volume*, termed as the "effective number of samples", which they calculate using a straightforward formula. This re-weighting scheme, thus, leverages that "effective number of samples" of each class to rebalance the loss; i.e. uses it as the class weight  $w_y$ . Overall, this approach was shown to be able to significantly improve performance, specifically on long-tailed datasets.

Based on that, we determine the class weights as follows:

$$\begin{aligned}v_y &:= (1 - \beta^{\#y^{level}}) / (1 - \beta) \\ w_y &:= \frac{v_y^{-1} |Y|}{\sum_{y_j \in Y} v_{y_j}^{-1}} := \frac{|Y|}{v_y \sum_{y_j \in Y} v_{y_j}^{-1}}\end{aligned}\quad (7.66)$$

where  $\beta \in [0, 1]$  is a hyper-parameter, and  $v_y$  is the effective number of samples, i.e. the sample volume;  $\beta = 0$  denotes no re-weighting and  $\beta \rightarrow 1$  approaches re-weighting relative to inverse class frequency (Cui et al., 2019). Note how we regard

the concept of (and determine the) sample volume "level-wise" for each hierarchy level; i.e. we use level-wise sample volumes.

### Take-Away

To **upweight uncommon class instances and downweight frequent class instances**, one approach is to employ traditional class weights, inversely proportional to their occurrence frequency. Alternatively, a more sophisticated strategy involves calculating class weights using the effective number of samples, as proposed by (Cui et al., 2019).

### Keywords

Class Weighting • Rare Class Upweighting • Common Class Downweighting  
• Effective Number of Samples

## Feature Independence

It is known that DNNs work better if input variables are decorrelated, as correlation can reduce the eccentricity of the error surface (LeCun et al., 2012). Similarly, we can wish to force the loss function to consider the feature independence (i.e. decorrelation) within embedding variables. This can be done easily by simply introducing an additional loss function that penalizes correlation between the features of the embedding vectors.

### Take-Away

We can improve the embeddings by lowering the correlation between the vector representations' feature components.

### Keywords

Embedding Enhancement • Correlation Reduction • Feature Independence

## Lipschitz Continuity & Spectral Normalization

Spectral normalization was proposed in Miyato et al., 2018 as a weight normalization technique to control the Lipschitz constant by constraining the spectral norm of layers. For a given real-valued function  $f$ , Lipschitz continuity is given when  $|f(x_1) - f(x_2)| \leq \mathcal{K}|x_1 - x_2|$ , where  $\mathcal{K} \in \mathbb{R}^+$  is the Lipschitz constant. The smallest value of  $\mathcal{K}$  is denoted by  $\|f\|_{Lip}$ . More formally,  $\|f\|_{Lip} := \sup_f(\sigma(W))$  for the spectral norm  $\sigma(W) := \max_{h:h \neq 0} \frac{\|Ah\|_2}{\|h\|_2} = \max_{h: \|h\|_2 \leq 1} \|Wh\|_2$  of a matrix  $W$ . The spectral norm  $\sigma(W)$  is equivalent to the largest singular value of  $W$ , which can be efficiently approximated using the *power iteration* method (Mises and Pollaczek-Geiringer, 1929). Regularization then occurs by transforming the weights using:

$$spec(W) := \frac{W}{\sigma(W)} \quad (7.67)$$

In the case of complex-valued networks, however, the spectral normalization must be rewritten so that it fits into the complex weight matrix. Therefore, given an

angle-weight matrix, we perform spectral normalization on both the imaginary and real matrix components:

$$\begin{aligned} W_{new} &:= \text{spec}(\cos(W)) + i \cdot \text{spec}(\sin(W)) \\ &= \frac{\cos(W)}{\sigma(\cos(W))} + \frac{i \cdot \sin(W)}{\sigma(\sin(W))} \end{aligned} \quad (7.68)$$

### Take-Away

We can also perform spectral regularization on complex bi-nonlinear networks by simply rescaling the real- and imaginary weight tensors by their respective spectral norms of the weight matrices.

### Keywords

Spectral Regularization • Complex-based Bi-Nonlinear Networks • Rescaling Weight Tensors • Spectral Norms • Weight Matrices

## Kernel Embeddings & Pooling Layer

Eventually, we have to transform our latent DNN *matrix* representation into a *vector* (i.e. embedding). We refer to this as the "pooling operation". Pooling operations, however, e.g. naively aggregating column vectors in a matrix, can have inherent limitations that may not be evident at first. While we do aim for embedding/pooling functions to be agnostic to the order of rows (there is no "ordering" of child nodes), we still want to ensure that distinct elements do not (or are not likely to) end up with identical representations. For instance, a mapping that chooses the maximum value in each column is independent of the order of rows *but* strongly injective, and so, easily produces identical representations for very "different" matrices that only coincide in their maximum value per column and have otherwise no similarity. In general, we want to prevent this. In other words: whenever we apply a pooling/embedding operation on a (deep latent) matrix, we want to ensure that the pooling operation, which converts it into a vector representation, captures as much information about that instance as it can; i.e. the aim is to minimize the loss of information here as much as possible, such that different instances only then have similar or equivalent representations, if their latent matrices are similar. In fact, imagine processing an entity through multiple deep neural layers to attain an "optimal" latent intermediate matrix representation; only to then apply a suboptimal and poor vector pooling operation on it that would completely "destroy" the entire progress and encoded knowledge. This would just totally undermine the quality and expressiveness of that good matrix representation we had before.

Also, conventional summation or averaging of columns, although commonly applied as pooling operations, do not effectively capture all the relationships between entries within a matrix, specifically not across columns. This means that the pooling operation does not explicitly take into account the associations between pairwise columns and rows, leading to a less expressive possible representation and, hence, to a potential loss of valuable information. Additionally, although the model's encoder and decoder layers are trained and optimized on matrices, training (i.e. the model weights; the forward function) is typically applied to each row (i.e., child) unless

special mechanisms like attention, etc. are used. Therefore, to address these limitations and explicitly consider pairwise aggregations of columns and rows, the pooling layer is key here and should, therefore, consider aggregations not only single-column-wise but also across multiple columns; i.e. pairwise. Overall, this underscores the importance and relevance of defining a *good* and pooling operation.

*Kernels* are a way to represent data. It allows us to map objects (i.e. non-linear observations) to a high dimensional space, e.g. to then allow for straightforward comparisons of intricate properties, perform classification, etc. The respective mapping is referred to as the *kernel function*. Given a kernel function  $k: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  for an input domain  $\mathcal{X}$  of size  $|U_\omega| \times m_\mathcal{X}$ , the kernel trick uses a "feature map"  $\varphi: \mathcal{X} \rightarrow \mathbb{H}$  over the reproducing kernel Hilbert space  $\mathbb{H}$  of dimensions  $n_\mathbb{H} \times m_\mathbb{H}$ :

$$k(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle_{\mathbb{H}} \quad (7.69)$$

where  $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ . Because the inner product is a measure of alignment, it automatically provides a measure of similarity between points. A kernel is called *characteristic* if the mapping is injective. Many different kernel functions exist (Muandet et al., 2016); a well known example of such is e.g. the standard *Gaussian kernel*  $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2)$ . A sufficient condition on the kernel  $K$  to ensure that  $D_K$  is a *valid kernel* (i.e. metric) is that  $K$  must be a positive definite kernel. Kernels can also be used to compute kernel *distances*  $D_K$ ; e.g. between sets of vectors or pairs of individual points (Phillips and Venkatasubramanian, 2011).

*Kernel methods* are computations that operate in a high-dimensional and implicit feature space *without* explicitly calculating the data's coordinates in that space. They instead compute e.g. the inner products of all data pairs in the feature space; which can then e.g. be used to compute kernel distances. Lastly, *kernel embeddings of distributions* (Muandet et al., 2016) is a non-parametric method in which a (probability) distribution is represented as a reproducing kernel Hilbert space element (Smola et al., 2007). Overall, in our context, we are interested in finding strong numeric vector representations of matrices with variable row sizes. A strong derivation of a one-dimensional embedding vector from a matrix (i.e. for the pooling operation) is crucial here, and we will propose a method based on the idea of kernel embeddings for feature matrices.

Following earlier notations,  $\mathbf{U}_\omega$  is a matrix representation of vector elements  $\vec{x}_i, \vec{x}_j \in U_\omega$ . Let  $h^*(\mathbf{U}_\omega) = h_k(h_{k-1}(\dots(h_1(\mathbf{U}_\omega))))$  denote the hidden state of size  $|U_\omega| \times m_k$  after forwarding  $\mathbf{U}_\omega$  through  $k$  layers;  $h^*(U_\omega)$  be the set of rows in  $h^*(\mathbf{U}_\omega)$ . A *pooling* operation is required to transform the hidden state matrix  $h^*(\mathbf{U}_\omega)$  to a vector. Let  $\mathbf{1}_n \in \{1\}^{|U_\omega|}$  be a vector full of ones, then  $z_{\mathbf{U}_\omega} := \mathbf{1}_{m_k} h^*(\mathbf{U}_\omega) = \sum_{i=1}^{|U_\omega|} h^*(x_i)$  naively provides a possible trivial pooling; the sum<sup>6</sup> of columns. However, such pooling only considers column-wise aggregations (no row-column correlations). *Joint distribution kernel embeddings* (JDE) (Muandet et al., 2017) can be used here to better encode the latent matrix information. Let  $\delta(i, j)$  be the indexing function:

$$\delta(i, j) := (i - 1)|U_\omega| + j \quad (7.70)$$

A full enumeration on the Cartesian product of  $h^*(U_\omega)$  can be pairwise identified by  $\{(h^*(\vec{x}_i), h^*(\vec{x}_j))_{\delta(i,j)} : 1 \leq i, j \leq |U_\omega|\}$ . And so, rewriting  $(h^*(\vec{x}_i)_{\delta(i,j)}, h^*(\vec{x}_j)_{\delta(i,j)}) := (h^*(\vec{x}_i), h^*(\vec{x}_j))_{\delta(i,j)}$  gives us the required format for JDEs, such that it holds:

<sup>6</sup>The *sum* is preferred over using the *average* or *maximum* for pooling aggregations in graphs since mean and maximum do not result in most expressive GNNs (Xu et al., 2018).

$$\mathbb{E}[h^*(U_\omega) \otimes h^*(U_\omega)] \triangleq \frac{1}{|U_\omega|^2} \sum_{j,i=1}^{|U_\omega|} h^*(\vec{x}_i)_{\delta(i,j)} \otimes h^*(\vec{x}_j)_{\delta(i,j)} \quad (7.71)$$

where  $\otimes$  is the tensor product. By equivalence of a tensor and a linear map, this can be interpreted as an uncentered cross-covariance operator (Song et al., 2009). When applied to matrices, the tensor product is also referred to as the Kronecker product (i.e., a special case of it). In fact, if we consider matrices as linear mappings, then the Kronecker product corresponds to the tensor product of linear mappings. However, the Kronecker product of matrices of size  $|U_\omega| \times |U_\omega|$  with itself produces a matrix with  $|U_\omega|^2 \times |U_\omega|^2$  dimensions. Its complexity, when calculated via a straightforward approach, would be  $O(|U_\omega|^4)$  due to the computation of all (new) matrix elements. This is prohibitively expensive for large matrices.

Instead, we propose a JDE-based *kernel*-embedding (KE) that scales with complexity  $O(|U_\omega|^2 + |U_\omega|)$ . Given  $h_i^{[j]}$  as the  $j$ -th element of  $h^*(\vec{x}_i)$ , it follows from the distributive law over the Kronecker product for a matrix representation  $\mathbf{U}_\omega$ , that:

$$\begin{aligned} h^*(\mathbf{U}_\omega) \otimes h^*(\mathbf{U}_\omega) &= \begin{bmatrix} h_1^{*(1)} & \cdots & h_1^{*[m_k]} \\ \vdots & \ddots & \vdots \\ h_{|U_\omega|}^{*(1)} & \cdots & h_{|U_\omega|}^{*[m_k]} \end{bmatrix} \otimes h^*(\mathbf{U}_\omega) \\ &= \begin{bmatrix} h_1^{*(1)} h^*(\mathbf{U}_\omega) & \cdots & h_1^{*[m_k]} h^*(\mathbf{U}_\omega) \\ \vdots & \ddots & \vdots \\ h_{|U_\omega|}^{*(1)} h^*(\mathbf{U}_\omega) & \cdots & h_{|U_\omega|}^{*[m_k]} h^*(\mathbf{U}_\omega) \end{bmatrix} \end{aligned} \quad (7.72)$$

Since  $h^*(\vec{x}_i) \otimes h^*(\vec{x}_j)$  denotes a sub-matrix *block*, it then follows directly that:

$$\begin{aligned} \mathbb{E}[h^*(\vec{x}_i) \otimes h^*(\vec{x}_j)] &:= \frac{1}{|U_\omega|^2} \sum_{i=1}^{|U_\omega|} \sum_{j=1}^{|U_\omega|} h^*(\vec{x}_i) \otimes h^*(\vec{x}_j) \\ &\triangleq \frac{1}{|U_\omega|^2} \text{block wise} \begin{bmatrix} h^*(\vec{x}_1) \otimes h^*(\vec{x}_1) & \cdots & h^*(\vec{x}_1) \otimes h^*(\vec{x}_{|U_\omega|}) \\ \vdots & \ddots & \vdots \\ h^*(\vec{x}_{|U_\omega|}) \otimes h^*(\vec{x}_1) & \cdots & h^*(\vec{x}_{|U_\omega|}) \otimes h^*(\vec{x}_{|U_\omega|}) \end{bmatrix} \\ &= \frac{1}{|U_\omega|^2} \text{block wise} \begin{bmatrix} h_{(1)}^{*[1]} h^*(\mathbf{U}_\omega) & \cdots & h_{(1)}^{*[m_k]} h^*(\mathbf{U}_\omega) \\ \vdots & \ddots & \vdots \\ h_{(|U_\omega|)}^{*[1]} h^*(\mathbf{U}_\omega) & \cdots & h_{(|U_\omega|)}^{*[m_k]} h^*(\mathbf{U}_\omega) \end{bmatrix} \\ &= \frac{1}{|U_\omega|^2} \text{block wise} \left( h^*(\mathbf{U}_\omega) \otimes \begin{bmatrix} h_{(1)}^{*[1]} & \cdots & h_{(1)}^{*[m_k]} \\ \vdots & \ddots & \vdots \\ h_{(|U_\omega|)}^{*[1]} & \cdots & h_{(|U_\omega|)}^{*[m_k]} \end{bmatrix} \right) \\ &= \frac{1}{|U_\omega|^2} \text{block wise} h^*(\mathbf{U}_\omega) \otimes h^*(\mathbf{U}_\omega) \\ &= \frac{h^*(\mathbf{U}_\omega)}{|U_\omega|^2} \sum h^*(\mathbf{U}_\omega) \end{aligned} \quad (7.73)$$

where  $\sum h^*(\mathbf{U}_\omega)$  denotes the matrix summation of (all) entries; notice how it is a scalar. Based on that, the final  $KE(\cdot)$  is defined over the kernel matrix  $K_{U_\omega}$  with

components  $K_{U_\omega}^{[i,j]} := h^*_{\cdot i} \sum h^*(\mathbf{U}_\omega)$ ; and so:

$$KE(U_\omega) := \frac{\mathbb{1}_{|U_\omega|} K_{U_\omega}}{|U_\omega|^2} = \frac{\sum_{i=1}^{|U_\omega|} K_{U_\omega}^{[i,:]}}{|U_\omega|^2} = \frac{\sum_{i=1}^{|U_\omega|} h^*(x_i) \sum h^*(\mathbf{U}_\omega)}{|U_\omega|^2} = \frac{\sum h^*(\mathbf{U}_\omega)}{|U_\omega|^2} z_{\mathbf{U}_\omega} \quad (7.74)$$

which returns a single vector representation for the latent matrix  $h^*(\mathbf{U}_\omega)$ ; where  $\mathbb{1}_{|U_\omega|} \in \{1\}^{|U_\omega|}$  is the one-vector of dimension  $|U_\omega|$ ; and  $z_{\mathbf{U}_\omega}$  the column-wise pooling aggregation (as mentioned earlier). Notice how the term  $\frac{\sum h^*(\mathbf{U}_\omega)}{|U_\omega|^2}$  represents a scalar (i.e. a scale factor). This, however, implies that  $h^*(\mathbf{U}_\omega)$  must not be standardized beforehand (e.g. via layer normalization), as:

$$\hat{h}^*(\mathbf{U}_\omega) := \frac{h^*(\mathbf{U}_\omega) - \mu_{h^*(\mathbf{U}_\omega)}}{\sigma_{h^*(\mathbf{U}_\omega)}} \implies \sum \hat{h}^*(\mathbf{U}_\omega) = 0 \implies KE(U_\omega) = \mathbf{0}_{|U_\omega|} \quad (7.75)$$

where  $\mu_{h^*(\mathbf{U}_\omega)}, \sigma_{h^*(\mathbf{U}_\omega)}$  are the mean and standard deviation of  $\mathbf{U}_\omega$ , and  $\mathbf{0}$  the zero vector. That is, standardizing  $KE(U_\omega)$  would nullify the *stretch* constant, which we denote by  $\lambda_\omega$ ; here being  $\frac{1}{|U_\omega|^2} \sum h^*(\mathbf{U}_\omega)$ . Moreover, standardization, unless explicitly accounted for, is not independent of the padding (it uses full  $|U_\omega|$ ) and can result in a zero division (i.e. zero standard deviation) in situations where  $h^*(\mathbf{U}_\omega) = 0 \vee \sum h^*(\mathbf{U}_\omega) = 0$ . Also, it does not guarantee a non-zero sum, i.e.  $\sum h^*(\mathbf{U}_\omega) \neq 0$ . As a result, we propose a different (yet more intricate) mapping for  $\lambda_\omega$ ; based on the same JDE principles as above. It involves the use of the strict lower triangular matrix  $L_{-1}$  (excluding the diagonal) of the *absolute* correlation matrix  $\text{Corr}[M_{\mathbf{U}_\omega}^T, M_{\mathbf{U}_\omega}]$  of the Gram matrix  $M_{\mathbf{U}_\omega} := h^*(\mathbf{U}_\omega)^T h^*(\mathbf{U}_\omega)$ ; mapped through a non-linear function:

$$\hat{z}_{\mathbf{U}_\omega}^* := \phi \left( \hat{z}_{\mathbf{U}} \underbrace{\frac{\sum |L_{-1}\{\text{Corr}[M_{\mathbf{U}_\omega}^T, M_{\mathbf{U}_\omega}]\}|}{\frac{m_k(m_k-1)}{2}}}_{\text{stretch factor: } \lambda_\omega} \right) = \phi \left( \hat{z}_{\mathbf{U}} \mathbb{E} \left[ |L_{-1}\{\text{Corr}[M_{\mathbf{U}_\omega}^T, M_{\mathbf{U}_\omega]}\}| \right] \right) \quad (7.76)$$

which is our final kernel embedding formula<sup>78</sup>;  $\phi$  is a non-linear function,  $\hat{z}_{\mathbf{U}}$  the standardization of  $\hat{z}_{\mathbf{U}_\omega}$ , and  $m_k$  the embedding dimension of  $h^*$ . Notice how  $\frac{m_k(m_k-1)}{2}$  is the number of elements in the strict lower-triangular matrix  $L_{-1}$ . A Gram matrix, in general, represents the inner products of all vector pairs in a given set. When the vectors are normalized, the Gram matrix is equivalent to the set's correlation matrix. However, since we showed earlier that we  $\mathbf{U}_\omega$  should *not* be normalized for the KE, we use the correlation over the Gram matrix, instead of computing the correlation over  $\mathbf{U}_\omega$ . The pooling layers are defined recursively in a bottom-up manner where the deepest levels of the hierarchy are passed first. Equation 7.76 has the benefit of being independent of padding rows within batches, considers cross-correlations, and empirically demonstrates good results. Most importantly, the pooling operation is row-invariant. Overall, the KE can be thought of applying sum-pooling and then multiplying an instance-specific scalar  $\lambda_\omega$  which encodes some "stretch" or "scaling"

<sup>78</sup>Depending on the specific dimensions of the features  $m_k$ , if the stretch factor values returned by  $\lambda_\omega = \mathbb{E}[|L_{-1}\{\text{Corr}[M_{\mathbf{U}_\omega}^T, M_{\mathbf{U}_\omega}]\}|]$  are too small (e.g. therefore negatively impacting training), we can set  $\lambda_\omega = 0.5(m_k)^2$  instead. This results in (slightly) larger stretch values and consequently smaller overall embedding magnitudes.

<sup>8</sup>Alternatively,  $\hat{z}_{\mathbf{U}_\omega}^* := \phi(\hat{z}_{\mathbf{U}} \alpha \lambda_\omega)$  might be used; where  $\alpha$  is a (learnable) scaling for  $\lambda_\omega$

information, where similarly correlated matrices have similar stretch factors.  $\hat{z}_{U_\omega}^*$  also shares principles concerning the expressive power of neighborhood aggregations in graph neural networks (Xu et al., 2018), where any injective multi-set function  $g$  may be represented as  $g(X) = \phi(\sum_{x \in X} f(x))$  for certain (non-linear) functions  $\phi, f$  (Xu et al., 2018).

Referring back to the original kernel embedding "trick", the representation obtained using Equation 7.73 has the advantage inherent in kernel methods while alleviating the disadvantage of increased computational expense, where for two input matrices of size  $(n_0 \times m_0), (n_1 \times m_1)$ , the Kronecker product (KP) would otherwise be defined over a  $(n_0 n_1 \times m_0 m_1)$  block matrix. We could also leverage the kernel similarity (i.e. kernel distance) between all instances given a batch of  $z$ -embeddings (Phillips and Venkatasubramanian, 2011), by training the model to decrease the kernel distance between equally labeled instances while maximizing the distance between unalike labeled ones (as in contrastive representation learning), e.g. using the **maximum mean discrepancy** between the average embeddings for all instances of the respective labels classes (see Section 7.5.9).

#### Take-Away

Instead of simply computing the average per column and losing valuable information, a different approach is to utilize a Kernel Embedding to encode "more" inter-correlated information into the embedding representations.

#### Keywords

Kernel Embedding • Information Preservation • Avoiding Information Loss

### 7.5.9 Efficient Contrastive Representation Learning

#### Negative Sampling-Based Contrastiveness

An entire contrastive correlation matrix needs  $|Z|^2$  pair-wise comparisons when there are  $|Z|$  many embeddings (i.e. samples). Because such matrix scales quadratically, this may be computationally costly, especially on bigger batch sizes. *Negative sampling* (Goldberg and Levy, 2014) employs a *negative* objective, specified exclusively on a portion of a corpus (i.e. batch), rather than the entire corpus.

Based on this idea, we can use an adaptation of this negative sampling technique to perform negative sampling-based optimization. Let  $p_{i,j}^+ \in P^+$  denote a positive pair  $(z_i, z_j)$  of instances with identical labels, and likewise, let  $p_{i,j}^- \in P_C^- \subset P^-$  be a negative pair; where  $P_C^-$  is a (random) subset of a desired cardinality of pure negative pairs  $P^-$ :

$$\mathcal{L}_{NS} = - \sum_{p_{i,j}^+ \in P^+} \log[\sigma(z_i^T, z_j)] - \sum_{p_{i,j}^- \in P_C^-} \log[\sigma(-z_i^T, z_j)] \quad (7.77)$$

where  $\sigma$  is the logistic *sigmoid* function. Alternatively, we can use (i.e. enforce) correlation instead:

$$\mathcal{L}'_{NS} = - \sum_{p_{i,j}^+ \in P^+} \text{Corr}[z_i, z_j] + \sum_{p_{i,j}^- \in P_C^-} \text{Corr}[z_i, z_j] \quad (7.78)$$

Although strictly spoken it is a different objective than the original contrastive loss, negative sampling is a form of noise contrastive estimation against random negative samples. Note that this approximation may not make much sense for extremely imbalanced data if most elements in a batch have the same label, i.e. when there are many more negative pairs than positive ones in the full correlation matrix. In case of unbalanced labels, we can e.g. simply select a subset of positive pairs  $P_C^+ \subset P^+$ , such that their cardinalities are about equal, i.e.  $|P_C^-| = |P_C^+|$ .

### Class-Label Based Contrastive Representation Learning

Further, to enforce contrastiveness, we can also train a model to correlate the embedding representations of equally labeled instances while imposing uncorrelation at instances of differing labels in a slightly different way. Given  $k$  entities, we can e.g. construct a correlation matrix  $\mathcal{R}^{k \times k}$  to accomplish this; similar to Zbontar et al., 2021; Hessel et al., 2021. The computation of an instance-based correlation matrix, however, grows quadratically with the number of entities. Also, one may wonder if imposing or preventing correlation between "instances" is the best approach.

Instead, we propose to first compute correlation instance-wise (i.e. we compute the correlation between pairwise instances), and after some training epochs, then over the mean representation vector of the corresponding class labels; i.e. in regards to the average embedding vector of instances with identical/equivalent class labels. Not only does this reduce the computational overhead considerably (we have fewer labels than instances; computation scales quadratically), but it also allows for a more flexible contrastive representation method that focuses on the overall contrastiveness between classes, rather than that of individual instances.

Let  $\bar{z}_y = \mathbb{E}[z_y]$  be the average embedding representation of a class  $y$  for embedding vectors  $z_y$  with a respective label  $y$ . Likewise,  $\bar{z}_{y|\kappa}$  denotes the average representation of the last  $\kappa$  (e.g. batch/train) iterations and  $\bar{Z}_{|\kappa}$  the embedding matrix of the mean representations of all classes and  $\mathbb{Y}$  the symmetric label instance matrix that has entries one if two instances have the same label and zero otherwise. Then, given  $k$  instances of an instance embedding matrix  $Z$  and  $c$  class labels:

$$\mathcal{L}_1 = \sum (\text{Corr}(Z, \bar{Z}_{y|\kappa}) - \mathbb{Y})^{\circ 2} = \sum \left( \begin{bmatrix} \text{corr}(z_1, \bar{z}_{y_1|\kappa}) & \cdots & \text{corr}(z_1, \bar{z}_{y_c|\kappa}) \\ \vdots & \ddots & \vdots \\ \text{corr}(z_k, \bar{z}_{y_1|\kappa}) & \cdots & \text{corr}(z_k, \bar{z}_{y_c|\kappa}) \end{bmatrix} - \mathbb{Y} \right)^{\circ 2} \quad (7.79)$$

defines a contrastiveness loss we can optimize for. Alternatively, another approach is to prioritize and focus on the correlation between the average representation of class labels:

$$\begin{aligned} \mathcal{L}_2 &= \mathbb{E} \left[ (\text{Corr}(\bar{Z}_{y|\kappa}) - \mathbb{Y})^{\circ 2} \right] = \mathbb{E} \left[ \left( \begin{bmatrix} 1 & \cdots & \text{corr}(\bar{z}_{y_1|\kappa}, \bar{z}_{y_c|\kappa}) \\ \vdots & \ddots & \vdots \\ \text{corr}(\bar{z}_{y_c|\kappa}, \bar{z}_{y_1|\kappa}) & \cdots & 1 \end{bmatrix} - \mathbb{1} \right)^{\circ 2} \right] \\ &= \frac{|\mathbb{Y}|(|\mathbb{Y}| - 1)}{2} \sum \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \text{corr}(\bar{z}_{y_c|\kappa}, \bar{z}_{y_1|\kappa})^2 & \cdots & 0 \end{bmatrix} = \frac{|\mathbb{Y}|(|\mathbb{Y}| - 1)}{2} \sum \mathbb{L}_{-1} \left[ \text{Corr}(\bar{Z}_{y|\kappa}) - \mathbb{Y} \right]^{\circ 2} \end{aligned} \quad (7.80)$$

where  $\mathbb{L}_{-1}(\cdot)$  is the strict lower triangular matrix (i.e. without the diagonal), and  $\mathbb{1}$  is the identity matrix. We use the strict lower-triangular matrix since the loss is symmetric; thus avoiding redundant operations that would otherwise only increase computational overhead. Overall, by optimizing the loss using the equation above, we do guarantee distinct representations for different labels, *but* do not (explicitly) enforce similar embeddings for instances with the same label. This is why we propose training in different stages, i.e. first instance-wise, then mean-embedding-wise, then label-wise.

### Mean-Teacher Average Class-Label Contrastiveness

Let  $\bar{Z}_{|\kappa}, \mathbb{1}$  be defined as above and, similarly,  $\hat{Z}_{|\kappa}$  the mean representation obtained using a "mean teacher network" as in 7.5.8. The loss:

$$\mathcal{L}_3 = \sum (\text{Corr}(\bar{Z}_{|\kappa}, \hat{Z}_{|\kappa}) - \mathbb{1}) \quad (7.81)$$

then provides a class contrastiveness loss whose computation complexity also scales with the number of classes/labels rather than the number of (batched) instances, essentially likewise allowing for a substantial computational performance improvement.

Hence, as we are using mean-teacher networks, we must further also "efficiently" maintain track of the average embeddings of all occurrences over all epochs and batches. Keeping track of all past and current embeddings during training and then just averaging all of them by demand is a brute-force strategy that is unacceptably slow, costly, and impractical. Instead, a cumulative moving average is a considerably better and more efficient approach. Overall, the vector representations are provided by the batches within every episode. A cumulative moving average in its basic form is only accurate if all random batches are of identical size, which is not always the case for the very final batch (e.g. for a batch size of four, a set of nine instances would yield two batches of size four and one batch of size one). However, we can simply modify the cumulative moving average to accommodate this (edge-) case. Let  $\bar{z}_n$  denote the running mean for representations  $z_j$  with  $1 \leq j \leq n$ , i.e.  $\bar{z}_n = \frac{1}{n} \sum_{j=1}^n z_j$ . For a new  $z_{n+1}$ , it follows trivially:

$$\bar{z}_{n+1} = \frac{z_{n+1} + \sum_{j=1}^n z_j}{n+1} = \frac{z_{n+1} + n\bar{z}_n}{n+1} = \bar{z}_n + \frac{z_{n+1} - \bar{z}_n}{n+1} \quad (7.82)$$

Similarly, for means of batches  $\bar{z}_p, \bar{z}'_q$ , with  $p, q$  respectively being the size of the batches, the mean  $\bar{z}_{p,q}$  is:

$$\bar{z}_{p,q} = \frac{p\bar{z}_p + q\bar{z}'_q}{p+q} \quad (7.83)$$

### Take-Away

To achieve an effective representation, we want to encode "similar" entities in a similar manner while ensuring that "dissimilar" objects are represented differently and far apart. Thus, we employ a contrastive representation strategy that encourages a strong correlation among similarly labeled instances and decorrelation among differently labeled instances.

**Keywords**

Contrastive Representation Strategy • Similarity & Dissimilarity • Correlation for Alike Instances • Decorrelation for Unalike Instances

### 7.5.10 Residual Information Preservation

The forwarding layers create intermediate latent representations by merging sub-embeddings and node features. To better ensure the preservation and consideration of information from the sub-embedding matrix, we can adopt principles of residual blocks (Zagoruyko and Komodakis, 2016) and then concatenate the Kernel-embedding of the sub-representation matrix to the node's intermediate representation (i.e. embedding). This concatenated information is then passed as a whole through an additional layer; yielding the final and residual node representation vector. Integrating residual preservation becomes particularly useful when dealing with nodes with many features; especially many categorical features of large dimensions. To enhance this process further, we can additionally employ an AE (especially for node levels with, again, high feature dimensions). The residual block idea is depicted in Figure 7.20 below:

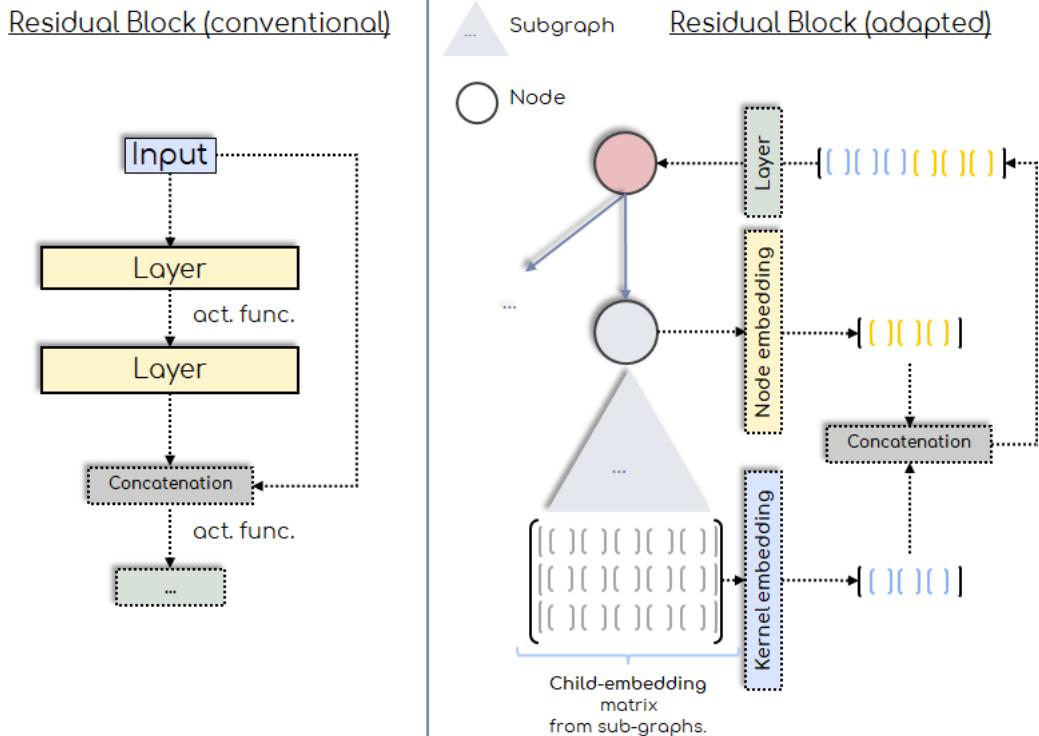


FIGURE 7.20: Preservation of sub-embedding information using a residual block-based approach.

### Fully Connected Residual Blocks and Basis Learning

The above concept of residual blocks may be extended not just to "within" or "successive" layers, but also "across" (all) hierarchy levels in the tree structure. Each level  $j$  generates a collection of embeddings (one for each node), which may be expressed as a matrix  $X_j$  (each row represents a node's vector representation). We can

then project each matrix  $X_j$  into a one-dimensional vector, again, using Kernel embeddings; yielding so a one-dimensional embedding for each level (each of which of possibly different dimensions). We can then combine these additional level-wise embeddings and pass them e.g. through a feed-forward layer (with a consistent output dimension) to eventually obtain a graph's final embedding (i.e. full representation); Figure 7.21 illustrates the main:

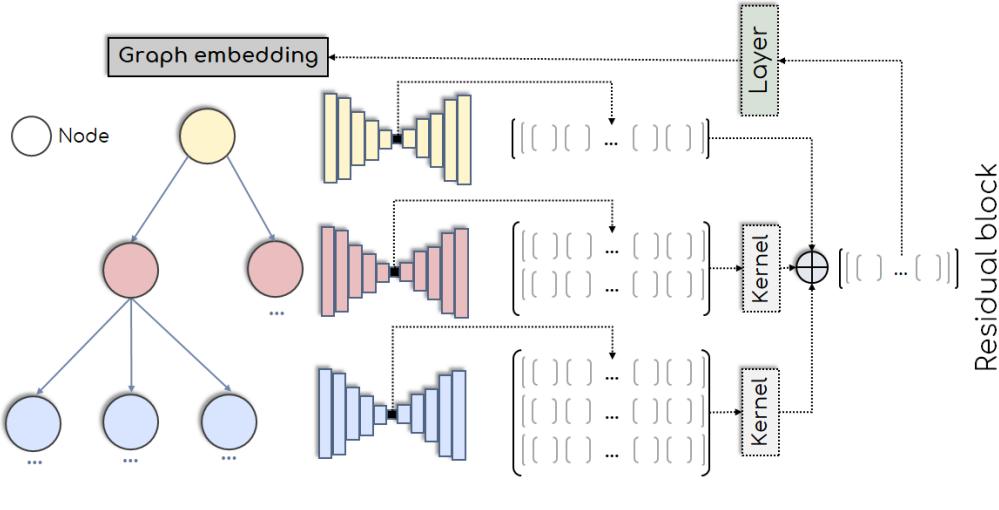


FIGURE 7.21: Preservation of sub-embedding information using a residual block-based approach over *all* levels.

Additionally, we may use fully and densely interconnected residual blocks to achieve a more expressive model architecture in terms of maintaining sub-embedding information (Figure 7.22):

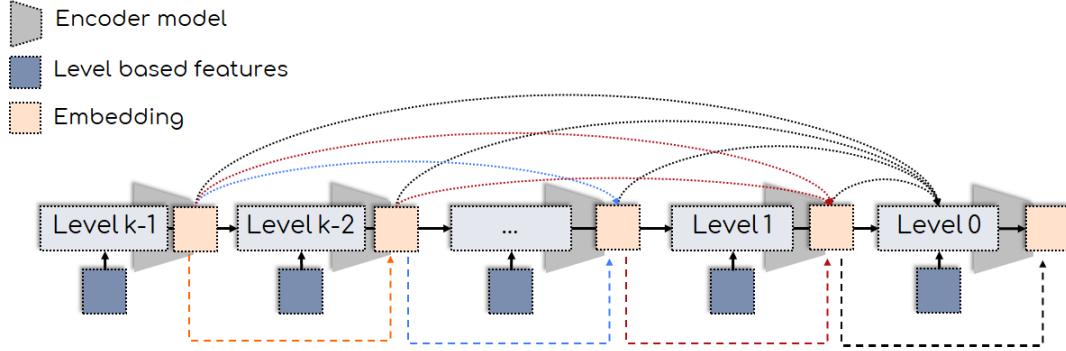


FIGURE 7.22: Full information preservation of sub-embeddings using a residual block "within" and "between" levels.

Yet, we can do even better: Li et al., 2015 proposed an interesting approach for sub-graph embeddings utilizing the concept of *virtual* nodes. These new so-called "virtual" nodes establish virtual connections with other nodes; and the virtual connections are then utilized to retrieve the node embeddings of all these interconnected nodes to ultimately yield a single combined embedding that represents or encapsulates the entire set of all those nodes (e.g. by further propagating the combined representation through additional layers, etc.). Specifically, in our hierarchical setting, we can introduce a virtual node for each hierarchy level, establishing so connections

to all nodes within that level. This ensures that the virtual embeddings are level-wise semantically meaningful, as the combined vector representation can capture correlations, patterns, etc. among *all* nodes at a given level; in particular, also among child nodes under different direct parent nodes. The embeddings of these virtual nodes are then further propagated upwards (again, via a residual layer block) to the root level. This is illustrated in Figure 7.23 below:

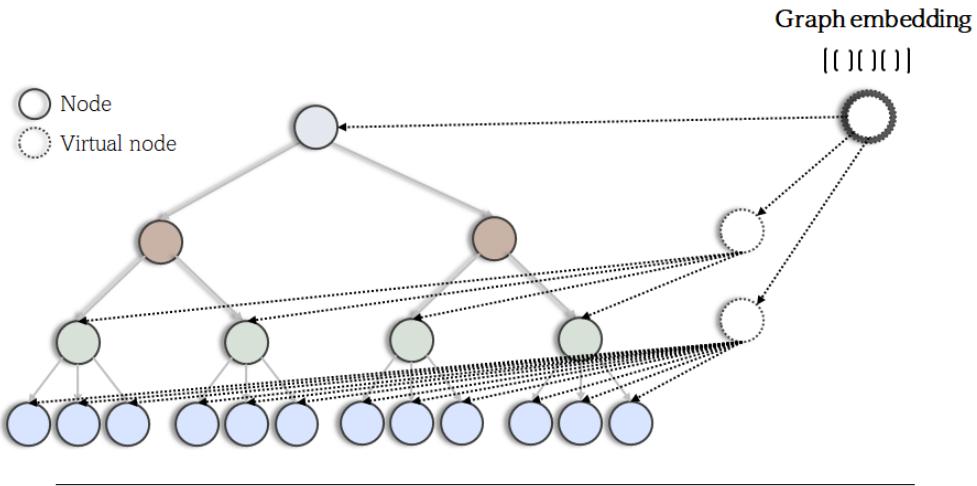


FIGURE 7.23: Fully connected residual blocks with virtual nodes.

A remaining question at this point, however, is *how* to (best) merge these virtual residual node embeddings together to produce a final graph embedding representation, especially when the hierarchy contains multiple sub-levels. We can also introduce importance scores (i.e. weights) for each level and aggregate the embeddings accordingly. This can be achieved through attention scores or assigning higher importance/scores to levels higher in the hierarchy. When manually setting the weights, however, it is unclear how these hyper-parameter weights should be optimally set for a given specific loss function or objective. We can also leverage and explore *basis learning*, which is utilized for graph learning on heterogeneous graphs; i.e. relational graph convolutional networks (Schlichtkrull et al., 2018).

Overall, notice how learning from residual block layers for aggregations of virtual node connections enables the model to learn from cross-correlations; since nodes are now inter-connected when training the model. This stands in contrast to e.g. an otherwise simplistic and raw averaging of sub-node embeddings, which would *not* ensure that the average of independent node representations is any meaningful. Instead, by training using virtual nodes and residual layer blocks, nodes without a common parent are now also implicitly trained together (via a common virtual node); and so, the final aggregated representation is meaningful and optimized (again, this is otherwise not achieved via an independent simple node averaging).

Assume  $n$  virtual nodes  $v_i$ , for  $1 \leq i \leq n$ ; each having an input matrix  $\mathbf{V}_i := \mathbf{U}_{v_i}$  (i.e.,  $\mathbf{V}_i$  is the child-matrix of the virtual node  $v_i$ ). Let  $\hat{z}_{\mathbf{V}_i}^*$  denote the respective KE (Equation 7.76).  $\hat{z}_{\mathbf{V}}^* := [\hat{z}_{\mathbf{V}_1}^* \oplus \dots \oplus \hat{z}_{\mathbf{V}_n}^*]$  be the concatenation of the vectors  $\hat{z}_{\mathbf{V}_i}^*$  of dimension  $m_{\mathbf{V}}$ . Given  $\sigma$  as a non-linear (activation) function and  $\tau$  as a linear feed-forward layer, the non-weighted (final) embedding would be  $\mathbf{z}_{\mathbf{V}} := \sigma(\tau(\hat{z}_{\mathbf{V}}^*))$ . Let  $\vec{\alpha}_v \in \mathbb{R}^n$  be importance scores of virtual nodes; and  $\vec{\alpha} \in \mathbb{R}^{m_{\mathbf{V}}}$ . It be  $\vec{\alpha}^{[m]} \equiv \vec{\alpha}_v^{[i]}$  iff index  $m$  refers to  $\mathbf{V}_i$  in  $\hat{z}_{\mathbf{V}}^*{^{[m]}}$ . Given  $\tau$ , custom (e.g. use-specific) weight-scores  $\vec{\alpha}$ , learnable importance weights  $\vec{w}_j$ , and a bias  $b_j$ , the  $j$ -th neuron's output is, therefore:

$$\begin{aligned}
\mathbf{z}_{\mathbf{V}}^{[j]} &\triangleq \sigma\left(b_j + \sum_{m=1}^{|\hat{z}_{\mathbf{V}}^*|} \alpha^{[m]} w_j^{[m]} \hat{z}_{\mathbf{V}}^{*[m]}\right) \\
&= \sigma\left(b_j + \sum_{i=1}^n \alpha_v^{[i]} \sum_{m=1}^{|\hat{z}_{\mathbf{V}_i}^*|} w_j^{[m+\sum_{a < i} |\hat{z}_{\mathbf{V}_a}|]} \hat{z}_{\mathbf{V}_i}^{*[m]}\right) \\
&\equiv \sigma\left(b_m + \sum_{j=1}^n \alpha_i \tau_i(\hat{z}_{\mathbf{V}_i})\right)
\end{aligned} \tag{7.84}$$

In other words, the final embedding vector  $\mathbf{z}_{\mathbf{V}}$  can be obtained by calculating the weighted sum of "individual" layer outputs  $\tau_i(\hat{z}_{\mathbf{V}_i})$ , *instead* of passing a weighted "concatenated" vector altogether through a linear layer. This is nice because it points out the option to easily substitute the linear layers  $\tau_i$  with any other feed-forward layer types or modules; demonstrating great flexibility in generating diverse aggregated weighted embedding representations.

Overall, the  $\alpha$  scores are normalized to first (1) add inter-weight dependencies; and second (2) ensure a mean-weight (i.e. importance) of one. Similar to Section 7.5.8, given  $n$  hierarchy levels, we perform the score weight normalization as follows:  $\hat{\alpha}_v^{[i]} := \frac{n\alpha_v^{[i]}}{\sum_{j=1}^n \alpha_v^{[j]}}$ , and so,  $\mathbb{E}[\hat{\alpha}_v^{[i]}] = 1$ . If  $\hat{\alpha}_v$  is chosen to be *trainable*, Equation 7.84 then shares similarities to basis learning (Schlichtkrull et al., 2018) in relational GNNs, as mentioned before, *without* having shared weight matrices.

Having introduced the concept of residual blocks *across* levels, RCs *within* levels can be defined analogously. Given  $U_{\omega} := \bigcup_{\theta_i \in \Omega_{\omega}} \{\vec{v}_{\theta_i,feat} \oplus \vec{v}_{\theta_i,sub}\}$ , its sub-embedding matrix be  $S_{\omega} := \bigcup_{\theta_i \in \Omega_{\omega}} \{\vec{v}_{\theta_i,sub}\}$ . Then, we define the within-level RC as:

$$\hat{z}_{(\mathbf{U},\mathbf{S})}^* := \phi_{\mathbf{U}}(\hat{z}_{\mathbf{U}}^* \lambda_{\mathbf{U}}) + \phi_{\mathbf{S}}(\hat{z}_{\mathbf{S}}^* \lambda_{\mathbf{S}}) \tag{7.85}$$

where  $\hat{z}_{\mathbf{U}}^*, \hat{z}_{\mathbf{S}}^*$  are the KE (Equation 7.76) for  $U_{\omega}, S_{\omega}$ , respectively;  $\phi_{\mathbf{U}}, \phi_{\mathbf{S}}$  are non-linear functions with equal output dimension;  $\lambda_{\mathbf{U}}, \lambda_{\mathbf{S}}$  are the kernel stretch factors.

### Take-Away

One approach to enhance the **preservation of sub-embedding information** is by employing residual (block-based) layers. Allowing the model to "learn" the importance of the residual components, or performing densely interconnected residual connections across all levels, are also options. In general, many residual methods and variations can be applied here in this context.

### Keywords

Information Preservation • Residual Layer Blocks • Dense Residuals

#### 7.5.11 Training & Inference

Because we are only interested in the final embedding vector during model prediction or -evaluation, we can deactivate and completely remove the decoder component, decreasing computation time and eliminating unnecessary computations overall. Moreover, during inference, the network undergoes slight modifications, such as the removal of dropout layers, the use of pre-computed statistical variables (e.g. batch normalization parameters), etc. Additionally, the inference hardware can also

differ and is often cheaper, as there is no need for intensive back-propagation; thus, allowing for specialized GPUs designed exclusively for inference tasks.

### Take-Away

For inference, decoder modules are redundant and can be ignored/removed.

### Keywords

Redundant Decoder Modules • Differences in Training vs. Inference

## 7.5.12 Full Model Composition

Based on the above extensions and concepts, we can combine them to establish a foundational model for a consolidated final NN layer architecture. An abstract high-level illustration is presented in Figure 7.24 below:

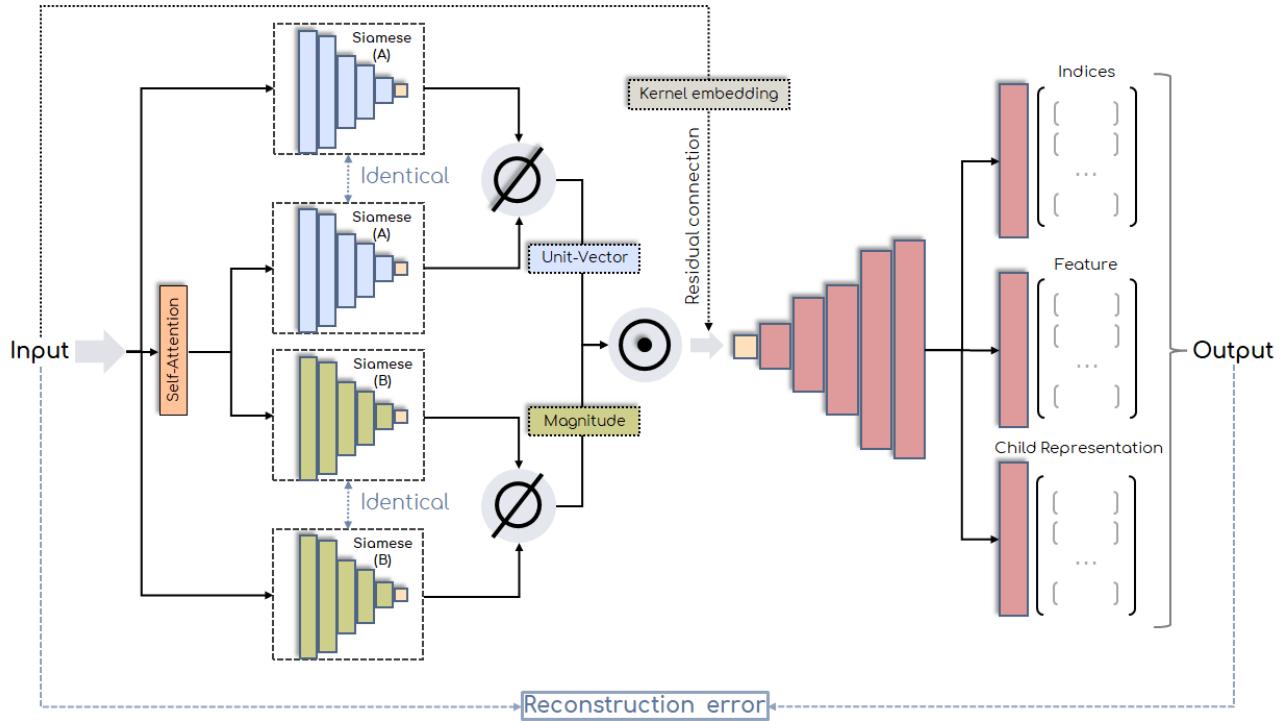


FIGURE 7.24: Forming a foundational unified hierarchical graph auto-encoder neural layer over a composition of multiple different concepts and ideas.

Given a model's input matrix  $U$ , it is first passed through a self-attention layer (e.g. scaled dot product); yielding the attention matrix  $U^\circ$ . Both matrices are then passed independently of each other through two distinct Siamese encoder modules and then pooled using kernel embeddings; outputting a total of four temporary intermediate output representation matrices  $Z_{A,1}, Z_{A,2}, Z_{B,1}$ , and  $Z_{B,2}$ , and each of which is then pooled using kernel embeddings into vectors  $\bar{z}_{A,1}, \bar{z}_{A,2}, \bar{z}_{B,1}$ , and  $\bar{z}_{B,2}$ .

To consolidate both (pairwise) Siamese outputs, we perform basic vector aggregation, for instance, by computing their means. Let  $\bar{z}_A := 0.5(\bar{z}_{A,1} + \bar{z}_{A,2})$  and

$\vec{z}_B := 0.5(\vec{z}_{B,1} + \vec{z}_{B,2}^\circ)$ . Following this, we normalize  $\vec{z}_A$  to achieve unit size and compute the norm for  $\vec{z}_B$ . The final embedding  $\vec{z}$  is, overall, given by:

$$\vec{z} := \frac{\vec{z}_A}{\|\vec{z}_A\|} \cdot \frac{\|\vec{z}_B\|}{\lambda_{\dim(\vec{z}_B)}} \quad (7.86)$$

where  $\lambda_{\dim(\vec{z}_B)}$  is the expected norm of a random Gaussian  $\mathcal{N}(0, \sigma I_{\dim(\vec{z}_B)})$  of dimension  $\dim(\vec{z}_B)$  with zero mean and standard deviation  $\sigma$ . Its expectation can be derived via the *Gamma*-function  $\Gamma$  (Chandrasekaran et al., 2012), i.e. the expected magnitude of the norm of a random Gaussian vector  $x$  is known to be:

$$\mathbb{E}[\|x\|_2] = \sigma \sqrt{2} \frac{\Gamma(\frac{\dim(\vec{x})+1}{2})}{\Gamma(\frac{\dim(\vec{x})}{2})} \quad (7.87)$$

and so we set  $\lambda_{\dim(\vec{z}_B)} := \mathbb{E}[\|x\|_2]$  accordingly.  $\Gamma(\epsilon)$  quickly leads to enormously large numbers (e.g., as can be easily seen for positive natural numbers  $n$ ,  $\Gamma(n) = (n-1)!$ ), and so,  $\Gamma(\epsilon)$  quickly leads to computational overflows. Using Sterling's formula, we can approximate  $\Gamma$ -quotients (Tricomi, Erdélyi, et al., 1951):

$$\frac{\Gamma(x+\beta)}{\Gamma(x+\alpha)} \approx x^{\beta-\alpha} \quad (7.88)$$

Setting  $\beta := \frac{\dim(\vec{z}_B)+1}{2}$  and  $\alpha := \frac{\dim(\vec{z}_B)}{2}$ , we know  $\gamma := \beta - \alpha = 0.5$ , and so for a standard Gaussian with  $\sigma = 1$ :

$$\sigma \sqrt{2} \frac{\Gamma(\frac{\dim(\vec{x})+1}{2})}{\Gamma(\frac{\dim(\vec{x})}{2})} \approx \sqrt{2} \dim(\vec{z}_B)^{0.5} = \sqrt{2 \dim(\vec{z}_B)} \quad (7.89)$$

However, following the inequality from Chandrasekaran et al., 2012, stating that:

$$\frac{\dim(\vec{z}_B)}{\sqrt{\dim(\vec{z}_B) + 1}} \leq \lambda_{\dim(\vec{z}_B)} \leq \sqrt{\dim(\vec{z}_B)} \quad (7.90)$$

provides a tight bound, given that we have  $\sqrt{\dim(\vec{z}_B)} < \sqrt{2 \dim(\vec{z}_B)}$ , we actually obtain a better "efficient" approximation using  $\lambda_{\dim(\vec{z}_B)} \approx \sqrt{\dim(\vec{z}_B)}$ .

Referring back to Equation 7.86, since we are multiplying a unit-vector times a scalar to obtain  $\vec{z}$ , this operation can be seen as *angle · magnitude* (note the similarity to complex-valued networks). In this context, the use of self-attention (applied to one of the inputs per Siamese network) ensures that the correlation (i.e., attention scores) among the row vectors of the input matrix are taken into account during computations; and its Siamese output is then averaged with the Siamese output of the non-correlated input to yield a robust (combined) intermediate representation. Again, this is done for each of both Siamese networks; yielding the matrices  $Z_A, Z_B$ .

For the inception-based decoder, the average  $Z := 0.5(Z_A \circ Z_B)$  is forwarded as input ( $\circ$  being the Hadamard product). The reconstruction loss is computed in regard to the original input  $U$ . Unless being on the root-level, the embedding  $\vec{z}$  is then further propagated upwards; this is all then done repeatedly (i.e. recursively) until reaching it.

**Take-Away**

The final model is formed by consolidating diverse techniques into a unified, advanced, single model. While the framework permits multiple modifications, the core architecture remains a hierarchical auto-encoder. However, incorporating additional techniques should be approached cautiously, as it may increase model complexity and training time.

**Keywords**

Main Model Integration • Single Architecture • Hierarchical Auto-Encoder

## 7.6 Concatenated Representation Learning

Consider now data entities that have multiple different data sources/views (e.g. hierarchical graphs, images, text documents, etc.); and for each of which we have representative embedding vectors/functions. The embedding representations can be merged by concatenating them into a unified larger single vector. Let  $\phi \in \Phi$  denote data instances from a set of entities, and be  $\Theta_0, \Theta_1, \dots, \Theta_k$  the  $k$  distinct data views. Thus, each  $\phi$  is associated with a corresponding  $\theta_i \in \Theta_i$ , where  $0 \leq i < k$ , and let  $\theta_i^\phi$  represent this association for a particular  $\phi$ . We write  $f_{\Theta_i}(\phi) \equiv f_{\Theta_i}(\theta_i^\phi)$  to denote the embedding function (producing a representation vector) of a given  $\phi$  for its respective  $\theta_i^\phi$ . The concatenated representation is so:

$$f_{\Theta}(\phi) := f_{\Theta_0}(\phi) \oplus f_{\Theta_1}(\phi) \dots \oplus f_{\Theta_k}(\phi) = [f_{\Theta_0}(\phi), f_{\Theta_1}(\phi), \dots, f_{\Theta_k}(\phi)] \quad (7.91)$$

where  $\oplus$  is the vector concatenation operation. We split the respective embedding set  $\{f_{\Theta}(\phi) : \phi \in \Phi\}$  as usual into training and test sets and feed it further into another DNN (Figure 7.25). Also, observe how the inputs now solely consist of standard numeric linearized vector values; i.e. concatenated vectors.

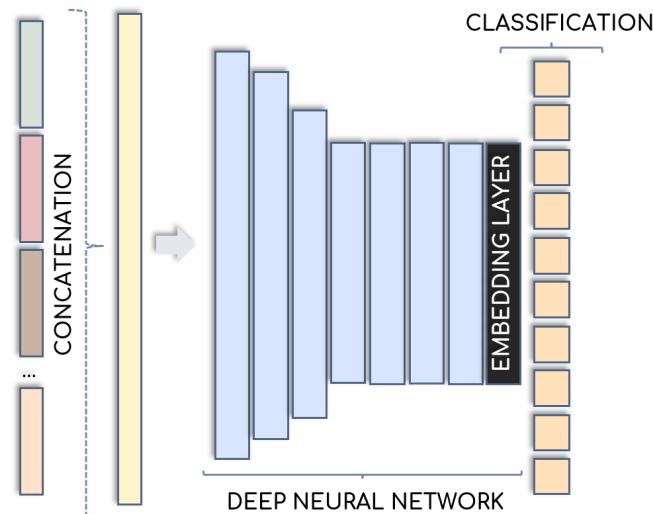


FIGURE 7.25: Illustration of feeding the concatenated vector representations into a further model; e.g. to learn a unified embedding.

Yet, it is crucial to maintain consistency and separation between training and test entities, ensuring that their intersection is always zero (i.e. disjoint). Specifically, for subsets  $\Phi_{train} \subset \Phi \supset \Phi_{test}$ , it must hold that  $\Phi_{train} \cap \Phi_{test} = \emptyset$ . Additionally, for all  $0 \leq i < k$ ,  $\Phi_{train}^{\Theta_i} \subset \Phi_{train} \supset \Phi_{test}^{\Theta_i}$  and  $\Phi_{test}^{\Theta_i} \subset \Phi_{test} \supset \Phi_{train}^{\Theta_i}$ . The same applies if an additional development set is introduced for hyper-parameter optimization.

By learning a final representation based on the concatenation of all individual data view embeddings, we can capture patterns across different representations and allow the model to jointly associate all data views (i.e. the concatenation); e.g. with respect to labels or other objective functions. If the embeddings are trained independently and then detached of gradients at the time of the concatenation, this also offers two further main advantages (over utilizing a single, end-to-end, large model):

(1) First, as data view representations can be trained separately, updating or introducing a new data view (e.g., in future time) only requires retraining the concatenated model along with the modified/new data view. There is no need to retrain the other models, i.e. the otherwise single very large model; and also across the entire dataset (i.e. other data views). This saves a lot of time and reduces the need for repetitive training (why go over data that has *not* changed?). Overall, this makes the model much more flexible and scalable. Additionally, since individual data views are trained independently, this further allows for parallel training.

(2) Data views may vary in size, complexity, and form. When dealing with a single larger model, training can encounter bottlenecks if, for instance, one data view is a shallow hierarchical structure while another is deeply layered. Calculating a single unified embedding requires, thus, waiting until all data views have propagated information upwards; and back-propagation also has to "wait" until information is transferred downwards (during training). Furthermore, because there is no gradient cut at the "concatenation level" and all sub-views are trained in correlation across their data structure and features (rather than solely via their final embedding representations), gradient calculation and optimization are far more complex, large and so computationally intensive; e.g. for hierarchical structures, gradients must flow down to the very bottom of all trees to all leaf nodes. In contrast, if the data views/embeddings are learned independently, gradients are only computed and calculated up to the concatenated vector (i.e. the gradient cut); *without* propagating gradients to the lowest (i.e. deepest) hierarchy levels.

### Take-Away

Instead of training a single extensive model incorporating all data views jointly, an alternative approach involves training individual models for each data view independently. The embeddings generated from these views can then be concatenated into a single concatenated vector; which is then passed through a last (non-hierarchical) model to produce the final embedding. This offers two advantages: (1) enhanced flexibility, and (2) improved computational performance.

### Keywords

Independent Data Training • Concatenated Embedding Representations

### 7.6.1 Handling Duplicate Instances

Entities may encompass instances of data views, e.g. hierarchical structures, where (sub-)trees or data parts are entirely identical. As we want a generalizable model that does not simply "fit" the most common data instance, it is imperative to remove duplicates inside corresponding data views. However, eliminating duplicated instances here is not a totally straightforward task, as entities might share identical instances only within specific sub-data views but not in combination with others; i.e. as a whole, they are not unique, but within individual data views, they are duplicates. Eliminating duplicate entities throughout the entire dataset, but selectively merely based on duplication within a specific subview, can not only greatly diminish the overall effective data size but also carries the risk of losing important information. On the other hand, not removing and retaining such duplicates can introduce issues like computational overhead, bias, overfitting, etc. Overall, when removing duplicates, careful consideration is essential, especially when splitting data into training and test sets, to ensure consistency between the two (this gets more complex when additionally using a development set). It is crucial to prevent any overlap between training and test sets across *all* data views to ultimately ensure an effective final model.

A solution is depicted in Figure 7.26. First, we identify duplicates within a data view containing instances  $\theta_i \in \Theta$ . Notice how  $\theta_i = \theta_j$  for  $i \neq j$  is possible (i.e. be aware of the difference between data and ids). We partition all instances accordingly into sets  $\Theta_{\perp}, \Theta_{\equiv}$ , respectively containing duplicate-free data ( $\Theta_{\perp}$ ) and duplicates ( $\Theta_{\equiv}$ ); with  $\{i : \theta_i \in \Theta_{\perp}\} \cap \{i : \theta_i \in \Theta_{\equiv}\} = \emptyset, \forall \theta_i, \theta_j \in \Theta_{\perp} : \theta_i \neq \theta_j, \Theta_{\equiv} \subset \Theta_{\perp}, \Theta_{\perp} \cup \Theta_{\equiv} = \Theta$ . Then, each  $\Theta_{\kappa}$ , for  $\kappa \in \{\perp, \equiv\}$ , is split into further train and test sets  $\Theta_{\kappa}^{train}, \Theta_{\kappa}^{test}$ . In other words, for each instance, if its inclusion in  $\Theta_{\perp}$  does not lead to duplicates, we add it there; otherwise, we add it to  $\Theta_{\equiv}$ . We then train the data view using only  $\Theta_{\perp}^{train}, \Theta_{\perp}^{test}$ . Since  $\Theta_{\equiv}$  is per definition a set of *duplicates*, we can "infer" the corresponding embeddings later over  $\Theta_{\perp}$  (for each duplicate in  $\Theta_{\equiv}$ , one correspondence in  $\Theta_{\perp}$  exists).

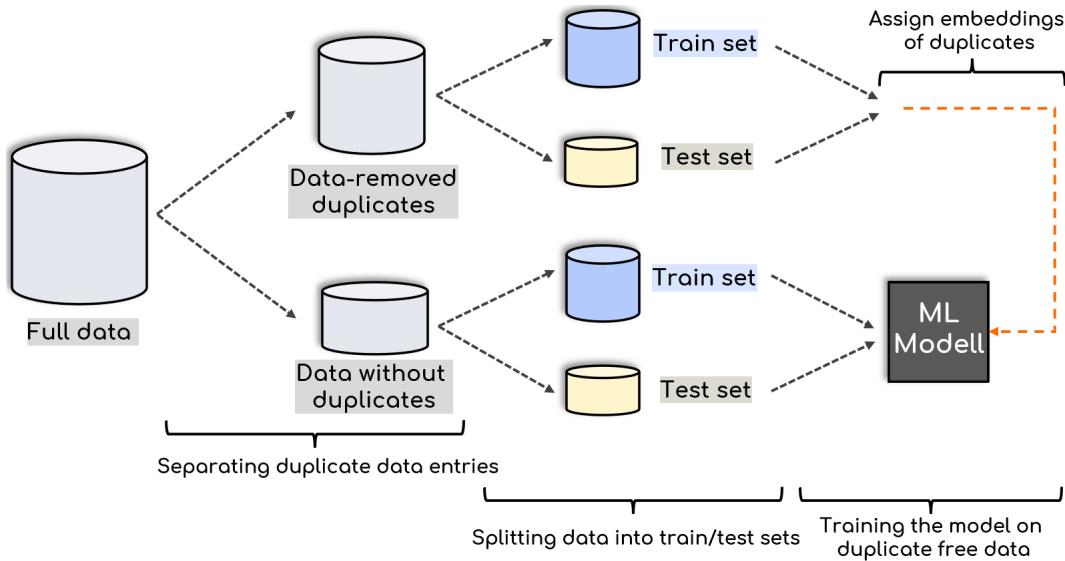


FIGURE 7.26: Suggested approach for separating data into train and test sets after identifying and removing duplicates to ensure data split consistency across different data views.

That is, when given multiple further data views, we maintain a consistent train and test split by defining the "entire" training set as  $\Theta_{\perp}^{train} \cup \Theta_{\equiv}^{train}$ ; and analogously, the test set as  $\Theta_{\perp}^{test} \cup \Theta_{\equiv}^{test}$ . Duplicates within each new data view are then also identified, respectively constrained to that training and test set partition, and removed accordingly; i.e. respecting the integrity of that partition. To establish an effective initial partition  $\Theta_{\perp}, \Theta_{\equiv}$ , the initial split should be performed on the data view containing the highest number of duplicates.

#### Take-Away

Duplicates are removed during training to avoid biasing the model, overfitting, unnecessary computations, etc. However, we must maintain consistency across *all* data views when splitting into train/test sets.

#### Keywords

Duplicate Data Removal • Bias Avoidance • Data Split Consistency

### 7.6.2 Memory Efficient Similarity Search

Let  $R$  be a set of reference texts, and  $T$  be a set of target texts. Let  $s : R \times T \rightarrow [0, 1]$  be a function that measures the similarity between a reference text and a target text; where 0 indicates no similarity and 1 indicates maximum similarity. We want to find the reference text  $r^* \in R$  such that  $r^* = \arg \max_{r \in R} \sum_{t \in T} s(r, t)$ . However, the naive brute-force search operation, which involves a nested loop, results in a prohibitively expensive time complexity of  $O(nm)$ ; including multiple model forward passes.

A somewhat better approach is to use a batching-based strategy, which forwards batches consisting of copies of the same and different reference texts together in a common batch through the score metric function. This allows the metric function to make use of vectorized operations to speed up the evaluation. The metric score outputs are then all assigned to the corresponding reference text and the mean score is calculated per reference text. Ultimately, the reference text with the highest average score is selected. However, the performance is directly dependent on the batch size; which could be small/limited depending on the available hardware resources.

A yet more efficient method consists of first forwarding all individual instances through the model in batches, caching the hidden model states (the output logits of the model), and then concatenating the hidden states. These concatenated states are then subsequently forwarded, again via batching, through the metric function. This avoids unnecessary repetitive computations and considerably speeds up score predictions. The different approaches are illustrated in Figure 7.27.

#### Take-Away

To efficiently assess the similarity between references and targets, we pass instances in batches through a model, cache hidden states, concatenate, and subsequently forward them (again in batches) through a metric function. This allows for a faster evaluation compared to a naive brute-force search.

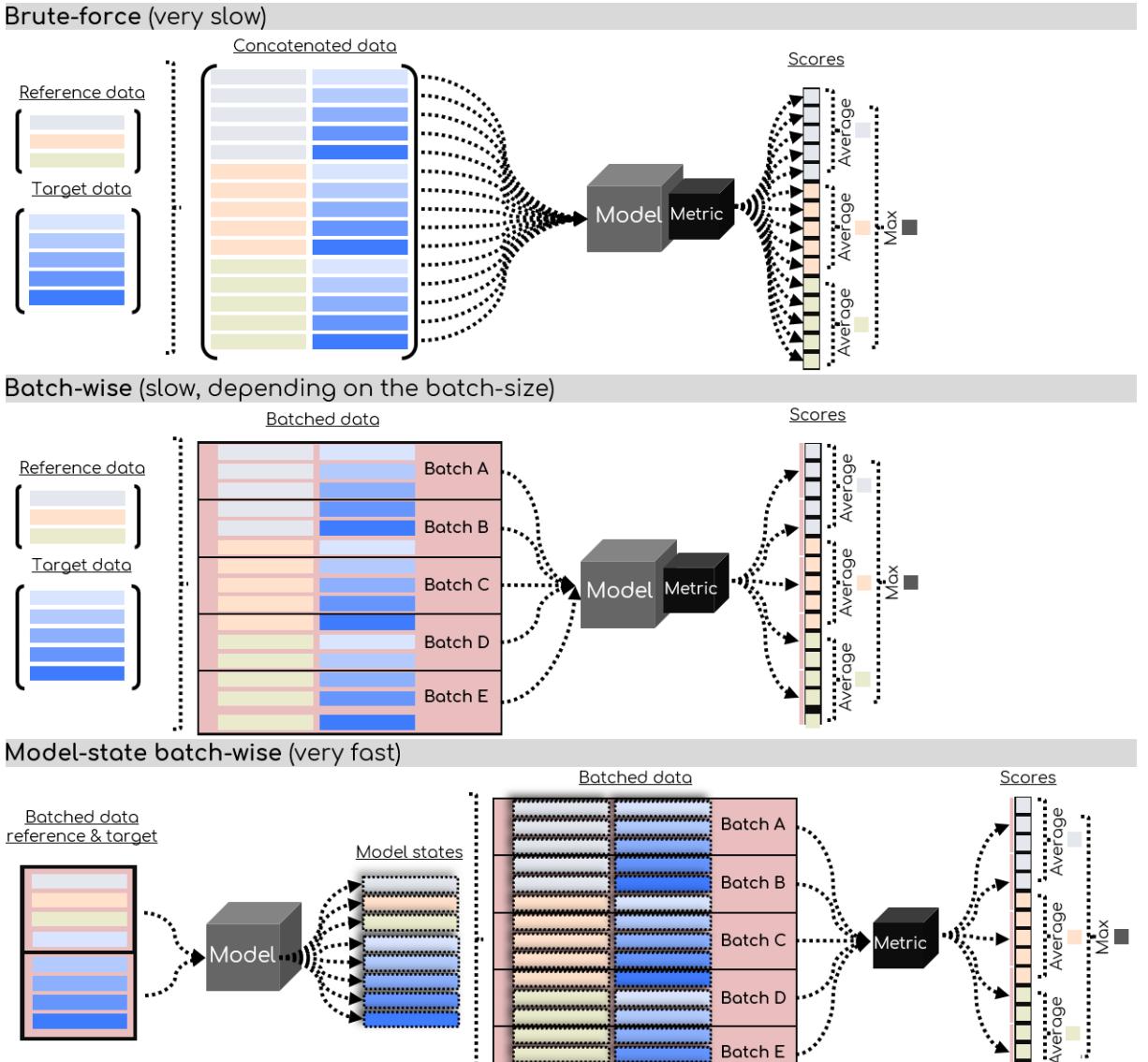


FIGURE 7.27: Similarity search using different methods.

### Keywords

Batching • Similar Reference Text Retrieval • Efficient Evaluation

### 7.6.3 Optimized Memory-Efficient Graph Batching

Graphs can greatly vary in size; i.e. they can have different numbers of nodes, levels, etc. When batching multiple graph instances together, the graphs are, thus, *padded* to fit the size of the largest graph in the batch. For example, consider a data set containing  $n$  graphs with an average of  $k$  nodes per graph, but a portion  $p$  of the data set contains extremely large graphs with an average of  $m$  nodes. If we randomly sample with a batch size of  $n/p$ , we can expect to have around one large graph per batch on average, thus, the padding dimension would be  $m$  (even though about all but one instance do not require padding at all). This is very inefficient because padding

consumes a considerable amount of memory; which can slow down training and increase computational costs. The greater the difference between  $n$  and  $m$ , the more severe this is. The total memory complexity is  $O(m(n/p))$ . In other words, naive batching is an inefficient use of computational resources, as padding might only be necessary to handle a *few* outliers; but is applied to *all* instances in the batch.

Hence, instead of storing all nodes as an explicit full padded matrix representation, we can use implicit representations, such as *nested* tensor matrices<sup>9</sup> (Figure 7.28). A nested tensor is a data structure that enables the efficient representation of data with non-uniform shapes. It is analogous to a regular tensor, i.e. a matrix, but differs in that not all dimensions (must) have uniform sizes; some dimensions can be irregular. This allows for more flexibility in representing data with varying shapes; and avoids (unnecessary) padding.

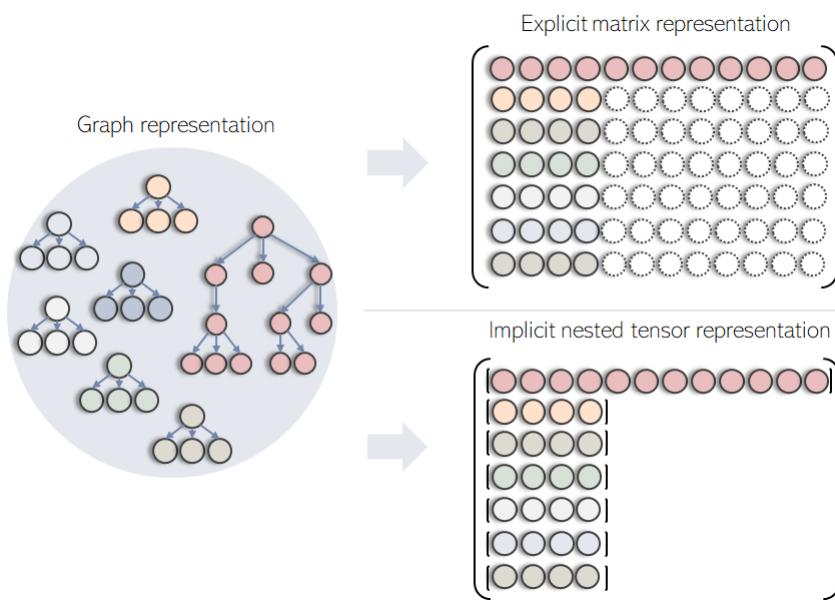


FIGURE 7.28: Using implicit nested matrix representations to avoid unnecessary and excessive padding in batch processing.

### Take-Away

Graphs vary in size with different numbers of nodes, levels, etc. When batching multiple graph instances, they are padded to match the size of the largest graph in the batch, resulting in inefficient memory usage. To address this, implicit representations like nested tensor matrices can be used, providing a more efficient and flexible approach that avoids unnecessary padding. This is particularly beneficial when dealing with greatly varying sizes and outliers.

### Keywords

Graphs • Batch Processing • Irregular Dimensions • Padding • Inefficiencies

<sup>9</sup>PyTorch currently supports Nested Tensors as a prototype data structure.

## 7.7 Manifold Data Representation and Clustering

Li et al., 2022 recently suggested in their work on *Neural Manifold Clustering and Embedding* (NMCE) that attaining manifold clustering with DNNs requires two fundamental ingredients: a domain-specific constraint that ensures the identification of the manifolds, and a learning algorithm for embedding each manifold to a linear subspace. NMCE follows three principles:

- The clustering and the representation must adhere to domain-specific constraints, such as local neighborhoods, local linear interpolation, or augmentation invariances.
- The embedding of a specific manifold must not collapse.
- The embeddings of manifolds must reside in different linear subspaces.

They achieve this by optimizing over two losses:

$$L_{NMCE}(X) := \underbrace{\lambda_0 MCR^2(X)}_{\text{learning algorithm}} + \underbrace{\lambda_1 Aug(X)}_{\text{identity constraint}} \quad (7.92)$$

where  $\lambda_{\{0,1\}}$  are weighting parameters,  $MCR^2$  is the *Maximum Coding Rate Reduction* objective (Yu et al., 2020) and  $Aug(x)$  a constraint function loss over data augmentation on  $x$ . Equation 9.7 above can be written into a more general form:

$$L(X) := \underbrace{\lambda_0 C(x)}_{\text{learning algorithm}} + \underbrace{\lambda_1 D(X)}_{\text{identity constraint}} \quad (7.93)$$

where  $C$  represents any objective for subspace feature learning.

Therefore, to ensure that the structure of the data is retained (i.e. not destroyed, e.g. by the DNN), alternatively to  $MCR^2$ , we can here also use an AE module<sup>10</sup> as the identity constraint loss function. As for the learning algorithm, we can use a classification loss; enhancing and encouraging so linear separability between individual classes/groups (i.e., manifolds). Therefore, our overall learning objective loss is:

$$L'(X) := \underbrace{\lambda_0 Class(x)}_{\text{learning algorithm}} + \underbrace{\lambda_1 AE(X)}_{\text{identity constraint}} \quad (7.94)$$

### 7.7.1 Taylor Approximation Smoothing

The  $k^{th}$ -order Taylor expansion around a point  $a$  of a  $k$ -differentiable function  $f(x)$  is the power series:

$$f(x) = \sum_{n=0}^k \frac{f^{(n)}(a)}{n!} (x - a)^n \quad (7.95)$$

where  $f^{(k)}(a)$  is the  $k^{th}$  derivative of  $f(a)$  and  $f^{(0)}(a) := f(a)$ . Generally, Taylor polynomial approximations to functions improve as  $k$  grows; although convergence is not always assured. If  $f(x)$  is infinitely differentiable, its Taylor series is:

<sup>10</sup>For learning diverse and discriminative representations, Yu et al., 2020 suggested using  $MCR^2$  over AEs due to the necessity for a robust and principled representation learning approach when handling intricate multi-modal data structures; overall, the authors argue that AEs can have certain challenges when trying to capture diverse subclass structures.

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (x-a)^n \quad (7.96)$$

Given *several* variables, the Taylor expansion of a function  $f(\mathbf{x})$  of  $n$  variables around the vector point  $\mathbf{a} \in \mathbb{R}^d$  in **multi-index notation** is:

$$f(\mathbf{x}) = \sum_{|\alpha| \geq 0} \frac{1}{\alpha!} D^\alpha f(\mathbf{a}) (\mathbf{x} - \mathbf{a})^\alpha \quad (7.97)$$

where  $\alpha = (\alpha_1, \dots, \alpha_n)$  is the multi-index,  $\alpha! = \alpha_1! \cdot \dots \cdot \alpha_n!$ ,  $|\alpha| = \alpha_1 + \dots + \alpha_n$ ,  $(\mathbf{x} - \mathbf{a})^\alpha = (w_1 - a_1)^{\alpha_1} \cdot \dots \cdot (w_n - a_n)^{\alpha_n}$ , and  $D^\alpha f(\mathbf{a})$  is the mixed partial derivative  $D^\alpha f(\mathbf{a}) = \frac{\partial^{|\alpha|} f}{\partial w_1^{\alpha_1} \dots \partial w_n^{\alpha_n}}(\mathbf{a})$ . Explicitly, this is equivalent to (Duistermaat et al., 2010):

$$f(x_1, \dots, x_d) = \sum_{n_1=0}^{\infty} \dots \sum_{n_d=0}^{\infty} \frac{\left( \frac{\partial^{n_1 + \dots + n_d} f}{\partial x_1^{n_1} \dots \partial x_d^{n_d}} \right) (a_1, \dots, a_d)}{n_1! \dots n_d!} (x_1 - a_1)^{n_1} \dots (x_d - a_d)^{n_d} \quad (7.98)$$

A DNN is essentially a *differentiable* mathematical function  $\mathbf{y} = f(\mathbf{x})$  that maps an input vector  $\mathbf{x}$  of dimension  $n$  to an output vector  $\mathbf{y}$ ; and so, this can also be approximated using a Taylor polynomial. The gradients of a DNN are typically computed with respect to the network's weights, but they can also be computed in regard to the input vector. Let  $\mathbf{x}_0$  be an input vector of a DNN function  $f$ . Then, Taylor's theorem can be applied to approximate  $f$  around a point  $\mathbf{a}$ , with components  $a^{[i]}$ ; for  $0 \leq i < n$ . The Jacobian matrix  $\mathbf{J}$ , whose components are denoted by  $[\mathbf{J}_{\mathbf{y}\mathbf{a}}]_{ji} := \frac{\partial y^{[j]}}{\partial a^{[i]}}$ , which is equivalent to  $\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial a^{[0]}} & \dots & \frac{\partial \mathbf{f}}{\partial a^{[k-1]}} \end{bmatrix}$ , defines the first-order Taylor approximation of the DNN around  $\mathbf{a}$ :

$$f(\mathbf{x}) \approx f(\mathbf{a}) + \mathbf{J}(\mathbf{x} - \mathbf{a}) + o(\|\mathbf{x} - \mathbf{a}\|), \quad (7.99)$$

where  $\|\cdot\|$  denotes the Euclidean norm and  $o(\|\mathbf{x} - \mathbf{a}\|)$  indicates an upper-bound that is not asymptotically tight, i.e. that the error of the approximation decreases faster than  $\mathbf{x} - \mathbf{a}$  as  $\mathbf{x}$  approaches  $\mathbf{a}$ . We denote first-order Taylor approximation of  $f$  for  $\mathbf{a}$  as  $f_{T_1}^{\mathbf{a}}$ .

Likewise, yet assuming twice differentiability on  $f$ , the second-order Taylor approximation could be used; which is computationally more demanding but takes into account the curvature of the function by using the symmetric Hessian matrix  $\mathbf{H}$  at  $\mathbf{x}$  with entries  $\mathbf{H}_{f_{i,j}} = \frac{\partial^2 f}{\partial x^{[i]} \partial x^{[j]}}$ . The approximation is:

$$f(\mathbf{x}) \approx f_{T_1}^{\mathbf{x}_0}(\mathbf{x}) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \mathbf{H} (\mathbf{x} - \mathbf{x}_0) \quad (7.100)$$

The trade-off for addressing the curvature with the second-order Taylor approximation is the substantially higher computational complexity (compared to the first-order Taylor approximation without considering curvature). In general, the quality of the Taylor approximation improves with decreasing  $\|\mathbf{x} - \mathbf{a}\|$  and deteriorates with increasing distance.

To describe and justify relationships between data, it is common to assume that the data follows at least one of the following distributions:

- **Smoothness:** Similar inputs should yield similar model outputs.
- **Continuity:** Close points are more likely to have the same label
- **Manifolds:** Data usually resides on latent manifolds of significantly smaller dimensions.
- **Clustering:** Data tends to concentrate into different clusters. Points inside a cluster are more likely to have the same label.

Combining these assumptions motivates the following thoughts:

**Supposition 1** *Instances on the same manifold are more likely to have characteristics that distinguish them from those on other manifolds and hence may form their own cluster.*

**Supposition 2** *Points that can be well approximated pairwise by predicting their position following the extrapolated local shape of the manifold around a particular point (e.g. linearly or through curvature) are more likely to be on the same manifold than points with pairwise high approximation errors.*

The idea of Supposition 2 is illustrated in Figure 7.29 below. It considers that points on the same manifold are more likely to have similar underlying characteristics, whereas points on separate manifolds do not. The white and red dots are on two different manifolds. When following the local manifold shape (e.g., linearly, through curvature, etc.) around a given point, the approximation between white points is substantially better than the approximation between red and white points. Their Euclidean distance may be small, but their manifold approximation is large. The accuracy of the approximation (obviously) depends on the smoothness and quality of the manifold landscape.

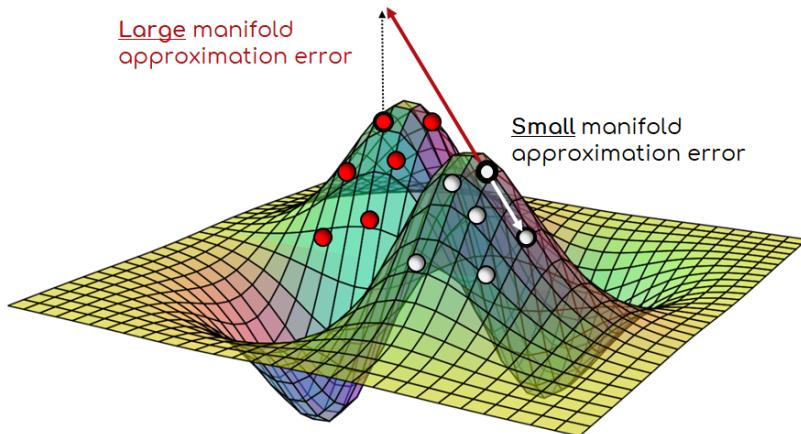


FIGURE 7.29: Illustration of the manifold approximation concept.

Thus, putting Suppositions 1 and 2 together leads to the following hypothesis:

**Hypothesis 1** *Subgroups within labeled data can be found by identifying the data's latent representations and building an artificial manifold space by forcing close points on the manifold of likewise labeled data to produce low manifold approximation errors while keeping those for points with dissimilar labels large.*

Because we can manipulate a DNN’s landscape by optimizing (i.e. decreasing) the prediction error over the Taylor expansion, smoothing out the landscape in this way can lead to changes in manifold sharpness (see Foret et al., 2020 for a thorough explanation of how smoothness influences generability). This allows us to further motivate the following observation:

**Assumption 1** *Optimizing for a low Taylor approximation error can impact the smoothness of the manifold landscape and, consequently, influence model performance.*

In other words, reducing the TA error could not only help us in identifying manifolds, but it may also change (i.e. improve) manifold smoothness and, hence, model performance. We propose the following Taylor-based manifold algorithm, which is divided into three main stages:

1. A warm-up phase in which a DNN model is trained using a classification loss on the embedding layer of an AE (together with reconstruction loss of the AE).
2. A smoothing phase in which a sub-module of the DNN further attempts to reduce the TA between the latent representations of equally labeled instances and their final output embeddings.
3. A clustering phase where we extract clusters based on the TA error.

Given an input  $x$ , let  $m$  be an AE model consisting of three (non-linear) parts: an encoder module  $m_\alpha$  with noise perturbed outputs, a fully connected intermediate module  $m_\beta$ , and a decoder module  $m_\gamma$ , such that a full model forward pass be defined as:

$$m(x) := m_\gamma \left( \underbrace{m_\beta \left( \underbrace{m_\alpha(x)}_{\text{encoder output } z_\alpha^x} + \underbrace{\eta^{[j]}}_{\text{noise } \in \mathcal{N}(0, \sigma^2)} \right)}_{\text{intermediate output } z_\beta^x} \right) \quad (7.101)$$

$$\underbrace{\qquad\qquad\qquad}_{\text{decoder output } z_\gamma^x}$$

where  $\eta^{[j]} := \mathcal{N}(0, \sigma^2)$  is noise in form a Gaussian with standard deviation  $\sigma > 0$ .  $z_\phi^x$  for  $\phi \in \{\alpha, \beta, \gamma\}$  be the respective latent vector representations of the sub-modules’ outputs, and  $w_\alpha^x := z_\alpha^x + \eta^{[j]}$ . We denote  $\mathcal{X}w_\alpha$  as the gradient-free (detached) copy of  $w_\alpha$ . Let further  $T_{\mathcal{X}w_\alpha^x}(m_\beta, \mathcal{X}w_\alpha^x)$  be the TA error matrix on  $m_\beta$  where gradients for back-propagation are computed *only* for the Jacobean  $J$  (or Hessian  $H$ ) over the network in regard to the gradient detached input vector  $\mathcal{X}w_\alpha$ . This forces the network to optimize the model’s approximation landscape rather than the latent vectors of the input points. Also, the gradient tree is smaller, so performing back-propagation is therefore computationally faster (i.e. has fewer operations). For notation, we write  $m_\delta(x) \triangleq m_\delta(z_\beta^x)$  as a separate classification module, defined for outputs  $z_\beta^x$  of the intermediate module  $m_\beta$ , which outputs scores for class labels.

In other words, our TA-based approach involves mapping input vectors into manifolds and then optimizing the TA based on those latent manifold representations. To retain the identity structure of the data, we utilize an auto-encoder, and to guide the learning algorithm from an unsupervised to a more supervised context, we incorporate a classification module. Our goal is to obtain suitable latent representations that encode information and semantics from both the labels and the data itself. By performing the TA over latent representations, we can ensure that the

distances between vectors are much smaller than if we were using raw input data. However, we must optimize the model's landscape rather than the vector coordinates by performing gradient-detach procedures. This, which we term *approximated latent spaces* (ATLAS), is summarized in the following pseudo-code (Algorithm 1):

---

**Algorithm 1:** Approximated latent spaces (ATLAS)

---

**Data:** Inputs  $x_i \in X$  with labels  $y_i \in Y$   
**Parameters:** Phase lengths  $p_1, p_2 \in \mathbb{N}$ ;  
**Initialize:** Encoder  $m_\alpha$ , Taylor model  $m_\beta$ , decoder  $m_\gamma$  and classification module  $m_\delta$ ;  
 $k \leftarrow 0$ ;  
// Warm-up phase  
**while**  $k < p_1$  **do**  
    Train sub-modules without Taylor approximation loss.;  
     $k \leftarrow k + 1$   
// Smoothing phase  
**while**  $k - p_1 < p_2$  **do**  
     $\epsilon \leftarrow 0$ ;  
    Extract embeddings  $\mathcal{W}w_\alpha, \mathcal{W}z_\beta$ ;  
**foreach**  $y := \text{label}$  **do**  
         $\epsilon \leftarrow \epsilon + \text{classification loss} + \text{auto-encoder loss}$ ;  
         $\Omega := \{x_i : y_i = \text{label}\}$   
        **foreach**  $x_i \in \Omega$  **do**  
             $\epsilon \leftarrow \epsilon + \sum_{\hat{x}_i \in \Omega} \|T_{w_\alpha^{x_i}}(m_\beta, w_\alpha^{\hat{x}_i}), \mathcal{W}w_\beta^{\hat{x}_i}\|_1$ ;  
        Back-propagate and optimize over  $\epsilon$ .;  
         $k \leftarrow k + 1$   
**finally**  
**foreach**  $y := \text{label}$  **do**  
         $\Omega := \{x_i : m_\delta(x_i) = \text{label}\}$   
         $Z := \{z_\alpha^{x_i} : x_i \in \Omega\}$ ;  
        Cluster  $\Omega$  using  $T_Z(m_\beta, Z)$  as metric;

---

To summarize, we follow a three-phase approach for training our model. In the first phase, we train the model on a pure classification and AE task. This helps to generate strong initial (i.e. representative) latent vectors which are then used for further Taylor optimization. If we try to optimize the model over the Taylor error from the start, it can result in highly unstable training. The warm-up phase ensures that the model possesses meaningful initial vectors and more stable sub-modules. Furthermore, as the TA lowers with increasing distance between points, we use classification to encourage the network to generate more compact representations for instances with identical labels.

In the second phase, we fine-tune the network landscape to produce good Taylor approximations with low errors. To prevent the intermediate model from producing "none-sense" and avoid a trivial collapsing solution (Li et al., 2022), we continue to train the TA loss alongside the classification and reconstruction losses. This also ensures that the model does not fully erase semantics previously encoded in the embeddings. Such an approach is similar to the NMCE loss (as in Equation 9.7), where a learning algorithm and an identity-preserving constraint are applied jointly:

$$\begin{aligned}
CLA(x_i, y_i) &:= \text{Classification loss of } x_i \text{ given label } y_i \\
AE(x_i) &:= \|x_i - m(x_i)\|_2 \\
TA(x_i) &:= \sum_{\hat{x}_i \in \Omega} \|T_{x_i}(m_\beta, z_{\hat{x}_i}), z_{\hat{x}_i}\|_1 \\
\epsilon_{x_i} &:= \underbrace{TA(x_i) + CLA(x_i, y_i)}_{\triangleq \text{learning algorithm}} + \underbrace{AE(x_i)}_{\triangleq \text{identity constraint}}
\end{aligned} \tag{7.102}$$

where  $\Omega := \{x_i : m_\delta(x_i) = \text{label}\}$ . Figure 7.30 illustrates the main concept of ATLAS graphically:

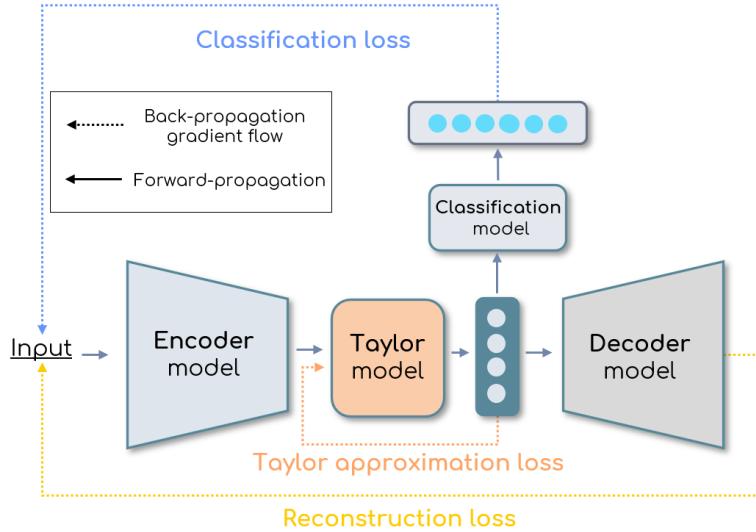


FIGURE 7.30: Illustration of the main architecture of ATLAS.

The third and last phase is to perform the actual clustering. Instead of utilizing the conventional Euclidean distance between vectors as a distance metric for clustering, we cluster based on the average pairwise TA error of points on model  $m_\beta$ . I.e., given  $x_i, x_j \in X$ , we define their approximation score  $s_{m_\beta}$  (higher is better) as:

$$\begin{aligned}
S_{x_i}(m_\beta, x_j) &:= \exp \left( \frac{\|T_{\hat{w}_\alpha^{x_i}}(m_\beta, \hat{w}_\alpha^{x_j})\|_1 - \mu_{T_{x_i}}}{\sigma_{T_{x_i}}} \right) \\
\hat{S}_{x_i}(m_\beta, x_j) &:= \frac{S_{x_i}(m_\beta, x_j)}{\sum_{x_k \in X} S_{x_i}(m_\beta, x_k)} \\
s_{m_\beta}(x_i, x_j) &:= -\min \left[ S_{x_i}(m_\beta, x_j), S_{x_j}(m_\beta, x_i) \right]
\end{aligned} \tag{7.103}$$

where  $\mu_{T_{x_i}}, \sigma_{T_{x_i}}$  are respectively the mean and standard deviation of the TA errors over a given instance  $x_i$  to all  $x_k \in X$ . We use the negative smallest Softmax scores from the normalized TA errors of instances as a clustering similarity metric. Note, that  $s_{x_i}(m_\beta, x_j)$  does not conform to a distance metric since it does not satisfy the triangle inequality; which is required for a *distance* function.

## Manifold Clustering

Because our TA-based score  $s_{m_\beta}(x_i, x_j)$  is a *similarity* metric rather than a distance metric. Note how not all clustering algorithms directly support similarities. Therefore, for clustering, we employ *Affinity propagation* (AP) (Frey and Dueck, 2007), which performs clustering using message passing between pairs until convergence. This works well for *similarity* metrics and determines the number of clusters automatically (but we could use any other similarity-based clustering method instead).

### Take-Away

Neural Manifold Clustering (NMCE) is a clustering technique that is specifically designed for DNNs that operate within complex geometric structures known as manifolds. The aim of NMCE is to group data points by mapping these manifolds onto linear subspaces, while also considering specific domain constraints. To achieve this, we can combine NMCE with Taylor approximation-based principles to identify points on the same manifold; which can help in subgroup retrieval/clustering. The idea involves training a DNN, then smoothing or optimizing latent manifolds using TAs, and finally performing clustering based on the TA errors.

### Keywords

Manifold Clustering • Deep Embeddings • Taylor Approximation Errors

## 7.8 Clustering Strategy

In a (semi-) supervised setting, given a set of embedding vectors, a naive technique would be to run a clustering algorithm simply on the entire vector set. However, we can do better. Instead, a more effective approach would be to perform  $n$  independent clusterings only on the embeddings with equal class prediction. If a classifier with  $n$  classes has already been trained, then using the predicted class labels (instead of the original labels) allows us to classify and cluster previously unseen instances that lack (coarse) labels; we simply predict (super-) labels using the classifier. In other words, we can focus the clustering group-wise on only those points that share the same prediction. This prevents the clustering algorithm from grouping instances with different labels together (assuming good accuracy for the classification module). In doing so, we effectively narrow down the clustering scope to only those points that have at least a common super/coarse label; trying to then find sub-clusters within each of those groups respectively. In fact, clustering could be done here repeatedly multiple times; i.e., after obtaining clusters within each class, we can further sub-cluster individual clusters by running a (possibly different) clustering algorithm to identify even deeper sub-clusters (Figure 7.31). Some implications are the following:

1. **Fine-grained clustering:** By performing independent clustering on embeddings with equal class predictions, clusters are intended to be more internally homogeneous with respect to their predicted labels. This can potentially reveal more subtle patterns within each predicted class.
2. **Utilizing classifier predictions:** Using predicted labels (instead of original labels) allows for clustering of new unseen instances without (coarse) labels.

3. **Mitigating the impact of incorrect groupings:** If the classifier employed to generate label predictions is accurate and effective, it helps to avoid severe incorrect clustering assignments. Grouping data points based on their predicted labels prevents instances with different (coarse-) labels but similar embeddings from being mistakenly clustered together.
4. **Hierarchical clustering:** Refining clusters through sub-clustering can expose deeper structures and relationships within each predicted class.
5. **Enhanced interpretability:** Visualizing hierarchical sub-cluster patterns within classes can aid exploration and decision-making, as well as provide more understandable insights into complex data structures and their distribution.

Again, it's important to note that the success of this method heavily relies on the accuracy of the initial classifier's predictions and the suitability of the clustering algorithms chosen for both the primary and sub-clustering tasks. Additionally, the computational complexity increases when doing multiple iterations.

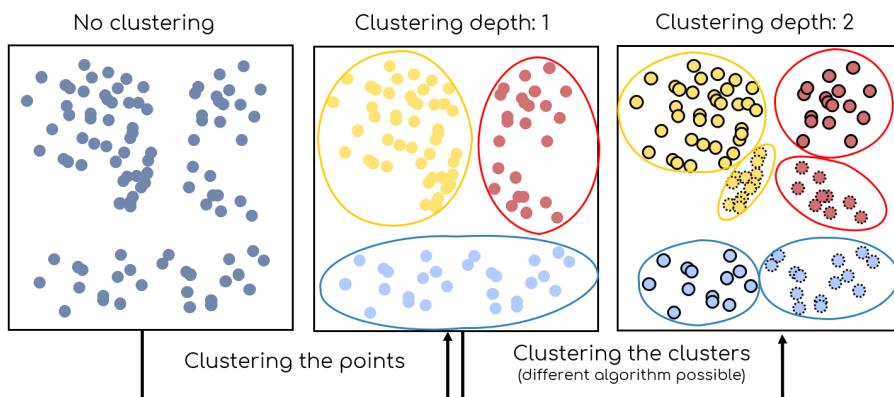


FIGURE 7.31: Performing sub-clustering on clusters to extract deeper and finer sub-patterns.

### Take-Away

One approach to cluster embedding vectors is by grouping only those instances respectively together with identical predicted class labels, rather than clustering the entire set indiscriminately. Clustering can also be then (re-) applied iteratively by sub-clustering identified clusters repeatedly.

### Keywords

Sub-Clustering • Sub-Pattern Identification • Label-Clustering • Refinement

## 7.9 Supervised Clustering Quality Evaluation

The B-Cubed ( $B^3$ ) algorithm (see Section 2.10.2, Amigó et al., 2009) can be used to numerically assess the quality of clustering assignments. However, it presupposes that the ground truth labels are exact and that there are no (relevant) sub-clusters inside groups of equally labeled objects. As an example, suppose we have  $m \cdot n$  many real, but to us unknown, total labels that can be divided into  $n$  sub-groups of pairs

of  $m$ , yielding so only  $n$  (observable) *coarse* classification labels; i.e. we only have an a priori knowledge of a subset of all ground-truth labels. We now want to discover (i.e. extract) all  $m \cdot n$  sub-clusters. Imagine we now have two clustering algorithms,  $A$  (which finds  $n$  coarse super-sets) and  $B$  (which finds all  $mn$  real labels/subsets). Although method  $B$  is obviously the one we want, the  $B^3$  algorithm would, however, favor algorithm  $A$ ; since  $B$  has a lower recall per cluster ("completeness", as in Figure ??). As a result, the constraint "completeness" can be problematic; meaning that the extrinsic  $B^3$  evaluation in its current form can be misleading when dealing with coarse and imprecise labels.

### 7.9.1 Background

Consider a data set  $X$  consisting of elements  $x_k$  with corresponding labels  $y_k \in Y$ . Let  $C_j \subset X$  be a cluster, indexed by  $j$ , such that the clusters are mutually disjoint, meaning  $\forall_{j,i} : C_i \cap C_j = \emptyset$ . The set of all clusters be  $C$ , and the union of all clusters is  $\bigcup_{C_j \in C} C_j = X$ , and so all elements belong to (exactly) one single cluster. Let the indicator function for two elements  $x_k, x_m \in X$  be denoted by  $\mathbf{1}_X : X \times X \rightarrow \{0, 1\}$ .  $\mathbf{1}_X$  returns a value of one iff both elements share the same label and belong to the same cluster; otherwise zero:

$$\mathbf{1}_X(x_k, x_m) := \begin{cases} 1 & \text{if } y_k = y_m \wedge \exists_{C_j \in C} : x_k, x_m \in C_j, \\ 0 & \text{otherwise.} \end{cases} \quad (7.104)$$

The  $B^3$  cluster score, which is based on precision, recall, and the  $F_\beta$ -score, is then defined via (Amigó et al., 2009) :

$$\begin{aligned} P(X) &:= \mathbb{E}_{x_k} [\mathbb{E}_{x_m | \exists_{C_j \in C} : x_k, x_m \in C_j} [\mathbf{1}_X(x_k, x_m)]] \\ R(X) &:= \mathbb{E}_{x_k} [\mathbb{E}_{x_m | y_k = y_m} [\mathbf{1}_X(x_k, x_m)]] \\ B^3(X) &:= F_\beta(X) \triangleq \frac{(1 + \beta^2)P(X)R(X)}{\beta^2 P(X) + R(X)} \end{aligned} \quad (7.105)$$

### 7.9.2 $\alpha$ Max-B-Cubed: A Supervised Metric for Addressing Uncertainty in Cluster Evaluation.

**Notice:** Our work and findings on  $\alpha$ Max-B-CUBED (presented in this thesis) have been publicly released on OpenReview-Venue (Guimerà Cuevas and Schmid, n.d.).

When attempting to identify sub-clusters within coarse ground-truth categories, the assumption that elements belonging to the same super-category should be grouped together ("completeness") can be problematic. Instead, we propose a modified evaluation metric called  $\alpha$ Max- $B^3$ . The core idea involves combining clusters based on their most frequent (i.e. occurring) label and consolidating them into new label-aggregated cluster sets.  $\alpha$ Max- $B^3$  is then calculated over a score weighting over the consolidated sets and the original clusters, utilizing a trade-off parameter  $\alpha$  for ground-truth uncertainty.

First, for a given cluster  $C_j$ , we define the most frequent label of all its elements as the cluster label:

$$\hat{y}_j := \operatorname{argmax}_{y \in Y} |\{x_k \in C_j : y_k = y\}| \quad (7.106)$$

Next, super-sets  $S_y$  are generated by merging clusters with equal max-pooled cluster labels  $\hat{y}$ :

$$S_y := \bigcup_{C_j \in C} \{x_k \in C_j : \hat{y}_j = y\} \triangleq \bigcup_{C_j \in C} \{x_k \in C_j : y = \operatorname{argmax}_{y' \in Y} |\{x_k \in C_j : y_k = y'\}|\} \quad (7.107)$$

A corresponding super-set for a particular label may be empty if that label never reaches the majority within any cluster; in which case it is just disregarded and ignored. In the unlikely event that  $\operatorname{argmax}$  is not unique, the cluster is simply *not* merged, left as it is, and does *not* count as a new super-set. Thus, given  $|Y|$  labels, at most  $|Y|$ -many new super-sets  $S_y$  are generated. The super-set generation is illustrated in Figure 7.32 below:

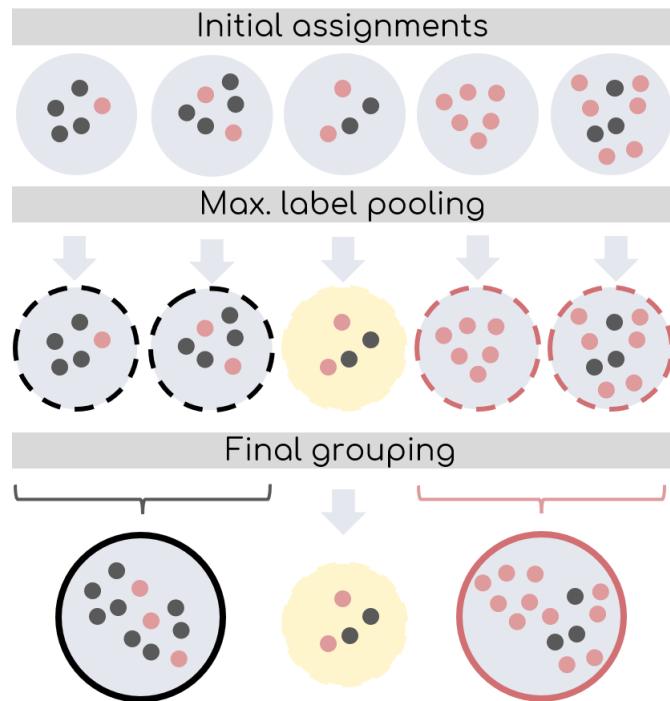


FIGURE 7.32: Conceptual illustration of the cluster aggregation and consolidation step for generating super-sets.

The final  $\alpha\text{Max-}B^3$  scores are then (a weighted average of) the  $B^3$  scores between the new (non-empty) super-sets  $\hat{S} := \{S_y : S_y \neq \emptyset\}$ , and the original cluster sets  $C$ . Let  $P_{[y]}(C_j)$  denote the  $B^3$  precision score of label  $y$  on elements in a cluster  $C$ , and  $R_{[y]}(C_j)$  respectively that of the recall. Let  $K$  be a set of cluster indices. We denote the weighted average precision score over multiple clusters as the expectation  $\mathbb{E}[\bigwedge_{k \in K} P_{[y]}(C_k)] := \sum_{k \in K} |C_k|_y P_{[y]}(C_k) / |\bigcup_{k \in K} C_k|_y$ ; the weighted recall score is defined analogously.

**Thm. 1** *Let  $C_i, C_j$  be disjoint clusters. Then:  $\mathbb{E}[P_{[y]}(C_i) \wedge P_{[y]}(C_j)] \geq P_{[y]}(C_i \cup C_j)$ .*

Theorem 1 (Proof B.2.1) shows that the precision score of the super-set is always less than or equal to the average precision score of the individual subsets. This implies a *monotonically decreasing* relationship in precision, which is crucial as it yields no fluctuations in score translations. A "decrease" in precision scores is not concerning because the actual cluster assignments of the elements remain unchanged; only

additional super-clusters are created. The evaluation of the cluster assignments is simply done on a different "scale"; whether scores are higher or lower in comparison is not directly relevant, but having a smooth monotonic transition is important. A stricter definition of Theorem 1 gives Proposition 1:

**Prop. 1**  $\mathbb{E}[P_{[y]}(C_i) \wedge P_{[y]}(C_j)] = P_{[y]}(C_i \cup C_j) \iff P_{[y]}(C_i) = P_{[y]}(C_j)$ .

This means that the average of the precision scores of two clusters for a given label  $y$  will only be equal to that of their super-cluster iff the precisions of both sub-clusters are the same (Proof B.2.2). It can be directly inferred from Theorem 1 and Proposition 1 that the translation is *strictly* monotonically decreasing when  $P_{[y]}(C_i) \neq P_{[y]}(C_j)$ , as we established both a "less equal" and an "equality" relationship. Without equality, Theorem 1 reduces to "strictly less" (Proof B.2.3).

**Cor. 1**  $\mathbb{E}[P_{[y]}(C_i) \wedge P_{[y]}(C_j)] > P_{[y]}(C_i \cup C_j) \iff P_{[y]}(C_i) \neq P_{[y]}(C_j)$ .

Analogously, let  $R_{[y]}(C)$  be recall scores of label  $y$  on elements in a cluster  $C$ .

**Thm. 2** *Let  $C_i, C_j$  be two disjoint clusters. Then:  $\mathbb{E}[R_{[y]}(C_i) \wedge R_{[y]}(C_j)] \leq R_{[y]}(C_i \cup C_j)$ .*

This, again, yields a monotonic score translation, but this time,  $B^3$  recall is higher on super-sets (Proof B.2.4). A stricter definition on Theorem 2 yields (Proof B.2.5):

**Prop. 2**  $\mathbb{E}[R_{[y]}(C_i) \wedge R_{[y]}(C_j)] = R_{[y]}(C_i \cup C_j) \iff R_{[y]}(C_i) = 0 \vee R_{[y]}(C_j) = 0$ .

And similarly, Theorem 2 and Proposition 2 imply the translation is *strictly* monotonically increasing whenever  $R_{[y]}(C_i) \neq 0 \wedge R_{[y]}(C_j) \neq 0$  (Proof B.2.6).

**Cor. 2**  $\mathbb{E}[R_{[y]}(C_i) \wedge R_{[y]}(C_j)] < R_{[y]}(C_i \cup C_j) \iff R_{[y]}(C_i) \neq 0 \wedge R_{[y]}(C_j) \neq 0$ .

Overall, the translation of the super-set scores (individually for precision and recall) is monotonic - respectively decreasing and increasing. However, the  $F_\beta$  measure, which is a composite function involving sums and products of both decreasing and increasing monotonic functions, does itself not necessarily exhibit monotonicity. Furthermore, although Theorems 1 & 2 refer pairwise to two clusters, they also still hold for any number of clusters as can be verified by repeatedly merging pairwise two clusters inductively (Proof B.2.7).

Let  $\mathbb{S}$  be the collection of all clusters. A super-set  $S_i \subset \mathbb{S}$  is the union of a specific list of clusters indexed by  $C_j$ , i.e.,  $S_i := \bigcup_{j \in \mathcal{J}} C_j$ , where  $\mathcal{J}$  represents the respective set of cluster indices. Each  $C_j$  is always assigned to exactly one  $S_i$ . The set of all identified super-clusters be denoted by  $\hat{\mathbb{S}}$ . Elements in  $\hat{\mathbb{S}}$  are mutually disjoint. The number of elements in a collection is denoted by  $|S_i|, |C_j|$ ; and  $|S_i|_y, |C_j|_y$  is the number of elements with label  $y$  in the respective collection;  $|y|$  be the total number of all elements with label  $y$ . Naively, one may now want to assign weights to the clusters based on their cardinality. Since each cluster index  $j$  is uniquely associated with a super-set index  $i$ , using  $j$  is sufficient to represent the weight. Let  $\eta^{[j]}$  be the weight assigned to cluster  $C_j$ , which naively is calculated as the ratio of the number of elements in  $C_j$  to the total number of elements in  $S_i$ . Using this naive cluster weight, we can derive the following equalities:

$$\eta^{[j]} := \frac{|C_j|}{|S_i|} = \frac{|C_j|}{|S_i|} \cdot \frac{|y||S_i|_y|C_j|_y}{|y||S_i|_y|C_j|_y} = \frac{\frac{|S_i|_y}{|S_i|}}{\frac{|C_j|_y}{|C_j|}} / \frac{\frac{|S_i|_y}{|y|}}{\frac{|C_j|_y}{|y|}} = \frac{P_{[y]}(S_i)}{P_{[y]}(C_j)} / \frac{R_{[y]}(S_i)}{R_{[y]}(C_j)} \quad (7.108)$$

Hence, we can express the ratio  $\eta^{[j]}$  of the cluster size to its super-set in terms of precision and recall. Theorems 1, 2 imply  $\mathbb{E}_{C_j \in S_i} \left[ \frac{P_{[y]}(C_j)}{P_{[y]}(S_i)} \right] \geq 1 \iff P_{[y]}(S_i) \leq \mathbb{E}[P_{[y]}(C_j)]$ ; and  $\frac{R_{[y]}(C_j)}{R_{[y]}(S_i)} \leq 1 \iff R_{[y]}(C_j) \leq R_{[y]}(S_i)$ ; given that  $|C_j|_y \leq |S_i|_y$  (Proof B.2.10); implying  $\mathbb{E}[\eta^{[j]}] \leq 1$  and the equivalence:

$$\begin{aligned} \mathbb{E}[\eta^{[j]}] &= \mathbb{E} \left[ \frac{P_{[y]}(S_i) R_{[y]}(C_j)}{P_{[y]}(C_j) R_{[y]}(S_i)} \right] = \mathbb{E} \left[ \frac{P_{[y]}(S_i)}{P_{[y]}(C_j)} \frac{\min[R_{[y]}(S_i), R_{[y]}(C_j)]}{\max[R_{[y]}(S_i), R_{[y]}(C_j)]} \right] \\ &= \mathbb{E} \left[ \frac{\min[P_{[y]}(S_i), P_{[y]}(C_j)]}{\max[P_{[y]}(S_i), P_{[y]}(C_j)]} \right] \cdot \frac{\min[R_{[y]}(S_i), R_{[y]}(C_j)]}{\max[R_{[y]}(S_i), R_{[y]}(C_j)]} \leq 1 \end{aligned} \quad (7.109)$$

Note that  $\mathbb{E}[\eta^{[j]}] \leq 1 \not\iff \forall_j : \eta^{[j]} \leq 1$ ; Let  $\alpha \in [0, 1]$  be a cluster-specific local uncertainty indicator. We can write a slightly modified non-monotonic  $\alpha$ -weighted expression of Equation 7.109 as:

$$\eta_\alpha^{[j]} = \frac{\min[P_{[y]}(S_i), \alpha P_{[y]}(C_j)]}{\max[P_{[y]}(S_i), \alpha P_{[y]}(C_j)]} \cdot \frac{\min[\alpha R_{[y]}(S_i), R_{[y]}(C_j)]}{\max[\alpha R_{[y]}(S_i), R_{[y]}(C_j)]} \quad (7.110)$$

where it holds  $0 \leq \eta_\alpha^{[j]} \leq 1$ , with  $\lim_{\alpha \rightarrow 0} \mathbb{E}[\eta_\alpha^{[j]}] = 0^+$  and  $\lim_{\alpha \rightarrow 1} \mathbb{E}[\eta_\alpha^{[j]}] = \mathbb{E}[\eta^{[j]}]^+$ ; both approaching from above. Therefore, it can be either  $\eta_\alpha^{[j]} \leq \mathbb{E}[\eta^{[j]}]$  or  $\eta_\alpha^{[j]} \geq \mathbb{E}[\eta^{[j]}]$  for different  $\alpha$ . Thus, non-monotonic (see Proof B.2.9). In fact, it is  $\min(p, \alpha q) = \max(p, \alpha q) \iff \alpha = p/q$ .

For arbitrary but *fixed* precision and recall scores, the function  $\eta_\alpha^{[j]}$  over  $\alpha$  has two "inversion points":  $p_1 := \frac{P_{[y]}(S_i)}{P_{[y]}(C_j)}$  and  $p_2 := \frac{R_{[y]}(C_j)}{R_{[y]}(S_i)}$ , and so  $\forall_{\min(p_1, p_2) \leq \alpha \leq \alpha' \leq \max(p_1, p_2)} : \eta_\alpha^{[j]} = \eta_{\alpha'}^{[j]}$ , which means that given fixed scores of precision and recall,  $\eta_\alpha^{[j]}$  has a function maxima plateau of score uncertainty in the interval  $[\min(p_1, p_2), \max(p_1, p_2)]$  (Proof B.2.11). The weight function  $\eta_\alpha$  over  $\alpha$  for sample values is plotted below in Figure 7.33. It reaches a maximum of one iff:  $\max[\eta_\alpha^{[j]}] = 1 \iff \frac{P_{[y]}(S_i)}{P_{[y]}(C_j)} = \frac{R_{[y]}(C_j)}{R_{[y]}(S_i)}$ .

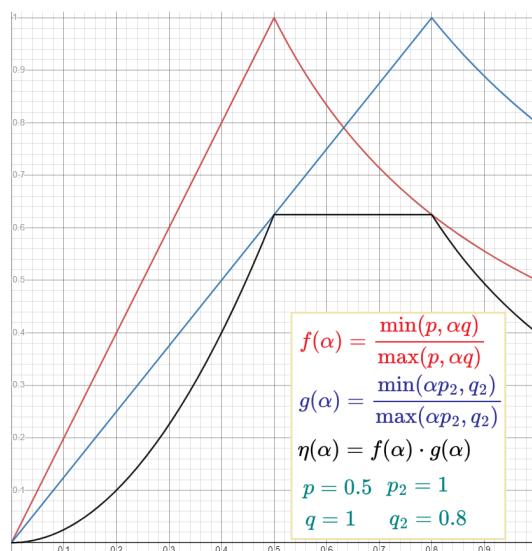


FIGURE 7.33: A graphical representation demonstrating the behavior of  $\eta_\alpha^{[j]}$  across varying values of  $\alpha$  using a fixed configuration of points as an example.

The local uncertainty score  $\eta_\alpha^{[j]}$  can be seen as the "super-set cluster importance" which weighs how important the score of the corresponding super-set is over the original cluster  $C_j$ . E.g., when the local uncertainty is zero, then  $\alpha$ -Max- $B^3$  is equivalent to  $B^3$ . Although  $\alpha$  can be set manually, we assume no prior knowledge of label uncertainty and set  $\alpha := \min(p_1, p_2)$  as the default choice for extracting the highest possible  $\eta_\alpha^{[j]}$  weight with minimal uncertainty.

Let  $Q \in \{C, S\}$ ,  $x_k \in C_j$ . We write  $Q_{k|q} \equiv x_k \in Q_q$  and  $k|j \equiv j$ . The final  $\alpha$ Max- $B^3$  score is the  $F_\beta$  score using:

$$\begin{aligned} P_\alpha(X) &\triangleq \frac{1}{|X|} \sum_{x_k \in X} \eta_\alpha^{[k|j]} P(S_{k|i}) + (1 - \eta_\alpha^{[k|j]}) P(C_{k|j}) \\ R_\alpha(X) &\triangleq \frac{1}{|X|} \sum_{x_k \in X} \eta_\alpha^{[k|j]} R(S_{k|i}) + (1 - \eta_\alpha^{[k|j]}) R(C_{k|j}) \end{aligned} \quad (7.111)$$

### 7.9.3 Extension to Imbalanced Data Sets

Dealing with imbalanced class distributions during evaluation can pose several problems. A straightforward averaging of  $F_\beta$  scores does *not* take into account the label imbalance. Similarly, conventional weighted averaging, which assigns weights inverse to frequency, may not be appropriate either. This is because it fails to consider the diminishing value of new data points as the number of items increases. The reason is, that when the volume of samples is large, there is an overlap in data information, and new data points are more likely to be close copies of existing ones. This is known as the *effective number of samples* (ENS) (Cui et al., 2019). The ENS measures the volume of a collection of  $n$  samples, given a hyper-parameter  $\delta \in [0, 1]$ , and is defined by the simple formula  $v_\delta(n) := (1 - \delta^n)/(1 - \delta)$ . Let  $|y_k|$  denote the frequency of label  $y_k \in Y$  in the data set. The normalized inverse weight is  $w_\delta(x_k) := v_\delta(|y_k|)^{-1} / \sum_{y_i} v_\delta(|y_i|)^{-1}$ ; and  $\delta$  is as a hyper-parameter. Cui et al., 2019 proposed a unified default value of  $\delta = (|X| - 1)/|X|$ ; which we will be adopting.  $w_\delta(\cdot)$  is then incorporated to define the balanced  $\alpha$ Max- $B_\delta^3$ :

$$\begin{aligned} P_\alpha^{[\delta]}(X) &\triangleq \sum_{x_k \in X} \frac{1}{|y_k|} w_\delta(x_k) P_\alpha(x_k) \\ R_\alpha^{[\delta]}(X) &\triangleq \sum_{x_k \in X} \frac{1}{|y_k|} w_\delta(x_k) R_\alpha(x_k) \end{aligned} \quad (7.112)$$

where  $P_\alpha(x_k) := \eta_\alpha^{[k|j]} P(S_{k|i}) + (1 - \eta_\alpha^{[k|j]}) P(C_{k|j})$  is the precision score of a given element  $x_k$ ;  $R_\alpha(x_k)$  is defined analogously for recall.

As mentioned in Cui et al., 2019,  $\delta = 0$  corresponds to no re-weighting, thus,  $\delta = 0 \implies P_\alpha^{[\delta]}(X) = P_\alpha(X) \wedge R_\alpha^{[\delta]}(X) = R_\alpha(X)$ ; while  $\delta \rightarrow 1$  approaches re-weighting on inverse class frequency. Theorems 1 and 2 remain true also for the imbalanced  $\delta$ -weighted version of precision  $P_{[y]}^\delta$  and recall  $R_{[y]}^\delta$ . The case  $\delta = 0$  is trivial, but it also holds for any fixed  $\delta \in [0, 1]$ , i.e.  $P_{[y]}^\delta(C_i) + P_{[y]}^\delta(C_j) \geq P_{[y]}^\delta(C_i \cup C_j)$  and  $R_{[y]}^\delta(C_i) + R_{[y]}^\delta(C_j) \leq R_{[y]}^\delta(C_i \cup C_j)$ . This becomes evident upon canceling out the weighting factor  $\delta$  on respectively both sides of the inequality (Proof B.2.8). The cancellation of the  $\eta_\alpha^{[k|j]}$  terms implies that any other weighting function can be used in place of the ENS and the theorems still hold true. Though essentially

an extension of the other, we have here independently addressed the  $\delta$ -weighted version separately for conceptual clarity.

### Take-Away

The B-Cubed evaluation method assumes that labels are exact and does not address the possibility of valid sub-labels. This can lead to wrong evaluations and model conclusions. We suggest a modified B-Cubed metric that considers predominant labels in clusters via super-classes and calculates a weighted loss over the "goodness" of original- and super-clusters. Specifically, we incorporate an uncertainty parameter  $\alpha$  that allows us to account for potential (and varying) degrees of uncertainty in the ground truth labels.

### Keywords

Clustering evaluation • B-Cubed • Label Uncertainty • Revised Metric

## 7.10 Automated Textual Description Generation of Clusters

We exploit the fact that images within a cluster have a semantic connection. This is due to the definition of clustering, which inherently groups "similar" items together. Our focus now is on generating descriptions for such clusters; not arbitrary groups of images. Our approach involves three stages:

- **Instance Captioning:** A ViED transformer model is first trained on annotated data and subsequently fine-tuned on a dataset of interest to enhance the captioning quality. If annotations are unavailable there, the pre-trained model is simply used to generate instance-wise captions (without fine-tuning).
- **Cluster Captioning:** The ViED is frozen. Image clusters are identified on the vision encoder's output. For each cluster, a pooling method is applied to obtain a respective cluster representation vector.
- **Caption Selection:** Multiple candidate sequences are generated using Top- $K$  (Fan, Lewis, and Dauphin, 2018) and Top- $P$  (Holtzman et al., 2019) sampling. Each is evaluated based on an automatic evaluation metric measuring the average text similarity to the cluster. The highest-scoring sequence is chosen as the most representative cluster caption.

### 7.10.1 Formal Specification

Let  $X = \{x_1, \dots, x_n\}$  denote a finite set comprising  $n$  instances, and  $C = \{C_1, \dots, C_m\}$  represent a set of  $m$  clusters. Each cluster  $C_j$  constitutes a non-empty subset of  $X$ , satisfying  $C_j \subseteq X$  and  $C_j \neq \emptyset$  for  $j = 1, \dots, m$ . We perform fine-tuning on an instance captioning architecture that involves an encoder model ( $E$ ) and a transformer decoder ( $D$ ), connected through cross-attention and equipped with a Language Model Head (LM-Head). This fine-tuning process utilizes the dataset  $(X, T)$ , where  $T = \{t_1, t_2, \dots, t_n\}$  represents the textual captions corresponding to instances in  $X$  and each caption  $t_n$  corresponds to instance  $x_n$ . The encoder  $E$  maps each instance  $x_i \in X$  to a latent representation  $z_i = E(x_i)$ , yielding a set of latent representations  $Z = \{z_1, z_2, \dots, z_n\}$ . Let  $Z_m$  respectively denote those for  $C_m$ . We denote the

text decoding strategy, which generates token sequences based on token probabilities, as  $\varrho$ . We represent text generation using decoder  $D$  with decoding strategy  $\varrho$  as  $D_{|\varrho}$ . Thus, the ViED model is expressed as  $D(E(x_i))$ , and its text generation is given by  $D_{|\varrho}(E(x_i))$ . Note that  $D(E(x_i)) \neq D_{|\varrho}(E(x_i))$ . The behavior of  $\varrho$  varies between training and inference due to the difference between the auto-regressive language modeling loss during training, which generates subsequent sequence words considering the preceding words in the ground-truth target caption, and the inference process, which generates words based on previously generated words in earlier steps. In our notation, we use  $\varrho$  to denote the inference process. To generate cluster descriptions, we leverage the diverse generation capacity of  $D_{|\varrho}$  and infer over a single cluster representation vector. We denote a vector pooling operation on the hidden states by  $\bullet[.]$ , so that  $\bullet[Z_m] =: v_m^\bullet$ . Since encoders  $E, E_{text}$  are frozen during training,  $v_m^\bullet$  is constant and can be cached. The entire forward pass of the model is:

$$\hat{y}_m = D_{|\varrho}(v_m^\bullet) \triangleq D_{|\varrho}(\bullet[E(C_m)]) \quad (7.113)$$

Multiple cluster captions (CCs) may be sampled as part of the decoding strategy  $\varrho$  over latent cluster representation  $v_m^\bullet$ . We denote the set of CCs by  $\hat{Y}_m = \{\hat{y}_m^{(1)}, \hat{y}_m^{(2)}, \dots, \hat{y}_m^{(k)}\}$ , where  $\hat{y}_m^{(j)}$  is the  $j$ -th generated candidate CC.

### 7.10.2 Contrastive Language-Image-Based Pooling

We employ a CLIP-based model (Radford et al., 2021) to align images with their corresponding textual descriptions. This alignment is achieved by training the model on vision and text encoder outputs generated from captions. The CLIP model comprises a text encoder, denoted as  $E_{text}$ , and two linear projection layers: one for the text encoder and another for the vision encoder  $E$ . During training, we optimize the CLIP InfoNCE loss (Oord, Li, and Vinyals, 2018) using (batches of) image embeddings  $Z$  and text embeddings  $Z^{text} := E_{text}(T)$ . This maximizes the alignment between matching image-text pairs while minimizing alignment between non-matching pairs. The projector layers are important as they ensure manifold compatibility, which allows us to calculate a CLIP score (Hessel et al., 2021) similarity matrix for hidden vision and text encoder states. Thus, given cluster  $C_m$ , for each instance's hidden state we can now compute the average CLIP score similarity to all captions in the cluster. This produces a CLIP score weighting matrix  $S$ , with a shape of  $n \times n$ . More formally,  $S$  is defined as  $S := \text{CLIP-SCORE}(Z_m^{text}, Z_m)$ .

#### CLIP Gradient-Guided Pooling Search

We can leverage the trained CLIP model to search for the most optimal pooling vector  $v_m^\bullet$  that yields the highest CLIP score when considering all the instance captions of a specific cluster  $C_m$ . In essence, we want the model to find/learn the most representative vector, parting from a basic mean-pooling approach. It's worth noting that text generation is inherently non-differentiable. In other words, we cannot directly optimize the generated captions given the visual embeddings<sup>11</sup>. This non-differentiability arises because the tokenization process disrupts the flow of gradients, preventing the update of the vision encoder parameters in response to the

<sup>11</sup>Categorical Reparameterization with Gumbel-Softmax (Jang, Gu, and Poole, 2016) allows back-propagation through discrete operations, but this does not provide an "easy" solution to the non-differentiability problem in our generative context. Also, the Straight-Through Estimator (STE) (Tieleman, Hinton, et al., 2012) is not suitable because it assumes the gradient of the non-differentiable operation is constant.

loss. Instead, we fine-tune the visual embeddings based on the descriptions. Consequently, optimization focuses on refining  $v_m^*$  with respect to  $T$  (Figure 7.34).

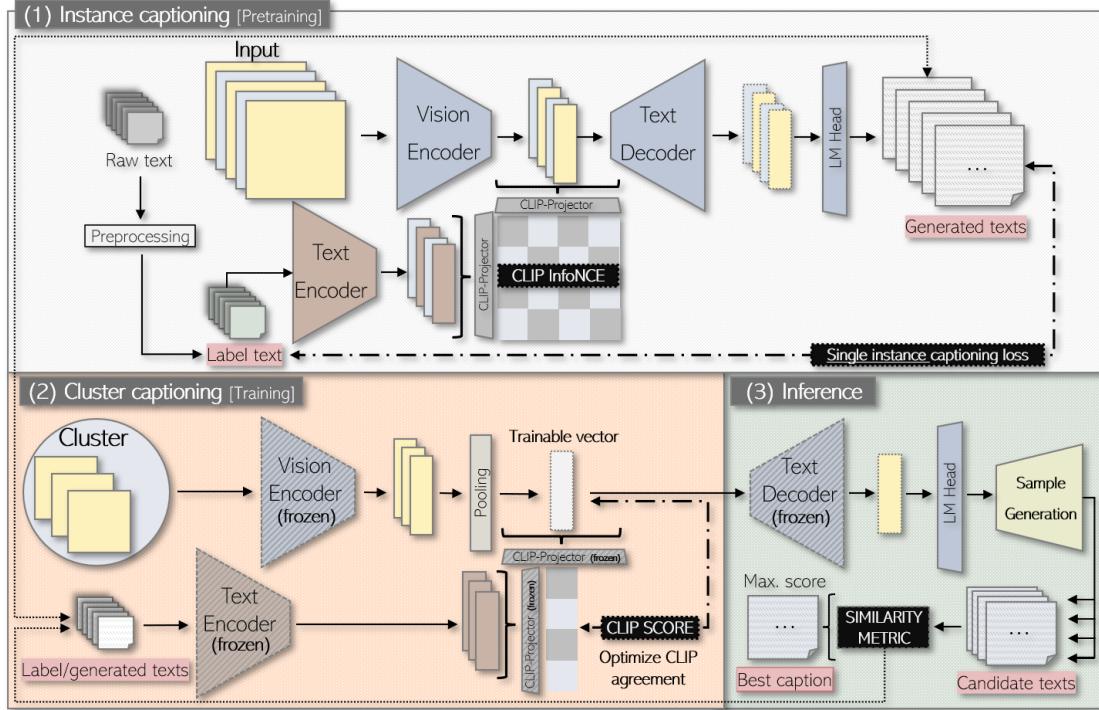


FIGURE 7.34: The full training architecture for generating CLIP-search-based cluster descriptions.

### CLIP Weighted Pooling

Rather than using a uniform weighted mean pooling approach, which weighs all instances equally, we can utilize CLIP similarity scores to assign greater weights to instances that demonstrate higher similarity to other instances within the cluster. These instances are often more representative, informative, or valuable. Instances with lower similarity may be too specific or less relevant to the cluster, and as such, should have a lesser influence on the embedding pooling process. We do this by calculating a score-weighted average of the hidden states from the vision and text encoders. The image-to-texts CLIP scores be respectively  $s := \sum_{i=0}^n S^{[i,:]}$ .  $\hat{s}$  is the normalized probability. The CLIP-weighted pooling then is:  $\sum_{i=1}^n \hat{s}^{[i]} Z_m^{[i]}$ ;  $n$  is the number of instances in  $C_m$ .

#### 7.10.3 Synthetic Boosting of Target Captions

To generate synthetic captions in the neighborhood of existing embeddings, we can create random normalized linear combinations of the hidden vision embeddings and forward them through the text decoder. This is similar to computing a random weighted mean of the embeddings. Various random sampling methods exist, such as Monte Carlo (Hammersley, 2013), Latin Hypercube (McKay, Beckman, and Conover, 2000), or Sobol (Sobol, 2001). This allows us to generate new captions by combining existing embeddings in a probabilistic manner with more control over the generation process and coverage of the embedding space. Let  $S_m$  be the synthetic embedding matrix of a tensor contraction between a probability distribution

$P$  of shape ‘sample-size’  $\times$  ‘embedding-size’ and the hidden embeddings  $Z_m$ ; with matrix components  $S_m^{[i,k]} = \sum_j P^{[i,j]} Z_m^{[j,k]}$ , where  $i, j, k$  represent the sample index, embedding index, and embedding dimension index, respectively. Textual neighborhood captions are then obtained via  $D_{|\varrho}(S_m)$ .

#### 7.10.4 Caption Selection

We evaluate the quality of generated cluster captions (CCs) using an automatic metric  $Q$ . It quantifies the similarity between each candidate CC  $\hat{y}_m^{(j)}$  and the set of reference targets for corresponding cluster instances  $T_m := \{t_i \in T : x_i \in C_m\}$ , yielding a scalar cluster score value  $q_m^{(j)} := Q(\hat{y}_m^{(j)}, T_m) \triangleq \mathbb{E}[Q(\hat{y}_m^{(j)}, t_i)]_{t_i \in T_m}$  that indicates the average semantic similarity between candidate CC  $\hat{y}_m^{(j)}$  and reference targets in set  $T_m$ . This requires that dataset  $X$  has corresponding textual instance-wise target annotations. If dataset  $X$  lacks captions (i.e., if  $T = \emptyset$ ), we can infer instance-wise captions using the pre-trained ViED model, provided it is domain-compatible to the dataset used for pre-trained, and set:

$$T \leftarrow \{D_{|\varrho}(E(x_1)), \dots, D_{|\varrho}(E(x_n))\} \quad (7.114)$$

The most appropriate CC representation is then the cluster caption with the highest average metric score ranked across all target candidates, identified by  $y_m^{(j)} := \operatorname{argmax}_{\hat{y}_m^{(j)}} Q(\hat{y}_m^{(j)}, T_m)$ . In other words, all candidate CCs are compared to the corresponding set of reference targets using an evaluation metric  $Q$ , and the one with the highest average score is selected as the most representative cluster description. The number of candidate descriptions to sample during decoding can be viewed as a hyperparameter  $k$ . In general, sampling decoding strategies select words typically proportional to word probabilities, and increasing  $k$  can lead to better and more diverse CCs; but at a higher computational cost.

#### 7.10.5 Quality Evaluation

The goal is to generate cluster captions that closely match textual descriptions of individual cluster images in terms of semantic similarity. We assess this using three key metrics: BLEURT (Sellam, Das, and Parikh, 2020), BERT-Score (**BERT-Score**), and BART-score (Yuan, Neubig, and Liu, 2021), collectively denoted as  $Q$ . These metrics measure text similarity by capturing semantic meaning in words and sentences. Additionally, BERT-Score and BART-Score provide insights via precision, recall, and F-scores. We excluded the CLIP score from the evaluation for fairness since we already optimized for it.

#### Take-Away

We can generate concise and descriptive captions for image clusters through a three-step process: (1) Training a ViED model for individual image captions. (2) Creating cluster representations and deriving captions. (3) Selecting the most representative cluster caption using text similarity metrics. Furthermore, we can integrate techniques like Contrastive Language-Image-based pooling and synthetic boosting of captions.

#### Keywords

Image • Clustering • ViED Transformer • Clusters Caption Generation

## 7.11 Prompt Manipulation on Large Language Models for Customized Text Summarization and Structured Analysis

A different powerful approach to summarize and explain clusters is by viewing the problem as a text-to-text generation problem (in contrast to image-to-text); using prompt engineering with large language models. For example, if we have identified a set of clusters, each with a textual description, we can pass the concatenation of all texts through a large language model, such as Llama2. In Llama2, the prompt and message template is given by:

```
<s>[INST] <<SYS>> {{ system_prompt }} </><SYS>>  
&{{user_msg}} [/INST] {{ model_answer }} </s>
```

where 'system\_prompt' is the model's internal prompt, 'user\_msg' is a given query text (e.g., user input), and 'model\_answer' is the output/response of the model. This allows us to leverage the power of large language models to generate informative and comprehensive summaries of clusters. The default prompt for Llama2 is:

#### Llama2 Default System Prompt (Touvron et al., 2023)

"You are a helpful, respectful, and honest assistant. Always answer as helpfully as possible, while being safe. Your answers should not include any harmful, unethical, racist, sexist, toxic, dangerous, or illegal content. Please ensure that your responses are socially unbiased and positive in nature. If a question does not make any sense, or is not factually coherent, explain why instead of answering something not correct. If you don't know the answer to a question, please don't share false information".

This template can be exploited by replacing the prompt with a summarization prompt, such as "What do these documents have in common?". However, unlike the approach described in Section 7.10, this method would not take raw inputs (e.g. images) into account and would only operate on textual documents. Nevertheless, we can combine both ideas by using a custom (vision) encoder and Llama2's text decoder to generate informative summaries of clusters.

#### Take-Away

Prompt-based engineering in large language models like Llama2 can be used to effectively summarize text clusters quickly. However, this method focuses solely on textual data and lacks integration with images or correlation between image representations and textual embeddings. Integrating a vision encoder with Llama2's text decoder could yield comprehensive summaries incorporating both textual and visual data within clusters.

#### Keywords

Prompt Engineering • Large Language Models • Textual Summarization

## 7.12 Determining Representative Textual Labels for Clustering Accurate Sensor Data with Inexact Annotations

We have a set of accurate sensor data and a set of annotations that may or may not be accurate. We are, thus, dealing with uncertainty here. Our task is to rank the annotations based on how well they represent the clusters in the sensor data. We need to determine which annotation is the best representation for a set of clustered sensor data while keeping in mind that the annotations may well be subjective, imprecise, wrong, irrelevant, or omit essential information, but likewise, also possibly very informative, accurate, and providing much extra information not included in the sensor data. The challenge is that the textual data is not aligned with the sensor data, so we need to account for strong uncertainty in the quality of the annotations.

We provide two separate methods for distinguishing "representative" annotations from "non-representative" annotations. The first method relies on the harmonic mean of three intuitive heuristics. The second one is based on the instance-wise lowest auto-regressive modeling loss over the cluster embedding.

### 7.12.1 Harmonic Mean of Intuitive Heuristics

We propose a method that uses the harmonic mean of three clustering score heuristics: sensor centrality, textual centrality, and textual cluster size. We will use the following observation and assumption: the closer a sensor data point is to the cluster center<sup>12</sup>, the more *likely* a corresponding textual comment is to be descriptive of the whole cluster since sensor data is accurate and reliable; and, thus, its respective cluster centroid is a good heuristic<sup>13</sup> of a point that is likely to represent the entire cluster. In contrast, if a sensor data point is an outlier or on the periphery of a cluster, the chance that its description is representative of the entire cluster is substantially lower; since a textual comment is based on its sensor data, so if a particular sensor data already has a lower quality, then the corresponding textual comment is also less likely to be of representative quality.

Let  $S, T$  here now denote sets of embeddings for sensor and textual data, respectively.  $(s_i, t_i)$  be a fixed pair of embedded sensor data  $s_i \in S$  with corresponding embedded textual data  $t_i \in T$ .  $C_j^{[S]} \subseteq S$  denotes the cluster assignment with index  $j$  on the sensor data  $S$ . For each cluster  $C_j^{[S]}$ , we perform clustering on the subset set  $T_j := \{t_i \in T : s_i \in C_j^{[S]}\}$  to obtain a set  $C^{[T_j]}$  of cluster assignments, where  $\forall_{C_k^{[T_j]} \in C^{[T_j]}} : C_k^{[T_j]} \subseteq T_j \subseteq T$ . We use  $z_j^{[S]}, z_k^{[T_j]}$  to denote the cluster centrality embedding vectors<sup>14</sup> (see Section 7.12.1) of  $C_j^{[S]}$  and  $C_k^{[T_j]}$ , respectively. Let the Euclidean distance score  $d_m^{[.]}(x)$  for  $C_m^{[.]}$  over the cluster centrality vector  $z_m^{[.]}$  be given by:

$$d_m^{[.]}(x_i) := \left\| x_i - z_m^{[.]} \right\|_2 \quad (7.115)$$

<sup>12</sup>This only applies to convex clusters. For non-convex clusters, we can use clustering membership strengths instead of centroids (see Section 7.12.1).

<sup>13</sup>Ultimately, factors like data distribution, cluster shape, density, dimensionality, and outliers influence the quality of a centroid in representing a cluster.

<sup>14</sup>We employ the term "centrality vectors" because not every clustering algorithm or cluster has a concept of centrality, i.e. a centroid. Clusters can be non-convex, and thus, the average of all cluster points might even lie outside the cluster set itself.

where  $\|\cdot\|_2$  denotes the (Euclidean) norm;  $[.]$  is either  $[C_j^{[S]}]$  or  $[C_k^{[T_j]}]$ ;  $m$  a corresponding cluster index.

We are interested in calculating probability scores, and the Softmax function is an intuitive and common solution here; well-known defined as:  $\sigma(x) = e^x / \sum_{x' \in \mathbf{X}} e^{x'}$  for an element  $x \in \mathbf{X}$ . To prevent numerical instability in computing the Softmax function where exponentials become very large due to large scores in the distance function, the "shift-trick" can be used, which subtracts the maximum score from each value. It maintains proportionality to the original function, since for any vector  $\mathbf{X}$  and any (single) arbitrary but fixed scalar constant  $c$ , the identity holds:

$$\sigma(x + c) = \frac{e^{x+c}}{\sum_{x' \in \mathbf{X}} e^{x'+c}} = \frac{e^x e^c}{\sum_{x' \in \mathbf{X}} e^{x'} e^c} = \frac{e^x e^c}{e^c \sum_{x' \in \mathbf{X}} e^{x'}} = \frac{e^x}{\sum_{x' \in \mathbf{X}} e^{x'}} = \sigma(x) \quad (7.116)$$

and therefore in particular  $\sigma(\mathbf{x} - \max[\mathbf{X}]) = \sigma(\mathbf{x})$ . This prevents overflow issues by ensuring that the largest exponential value is zero. The shift-trick for Softmax, however, can still result in large negative values, but this does not cause numerical instability to the same exponential extent. Overall,  $\sigma(\mathbf{x} - \max[\mathbf{X}])$  will always return a value between 0 and 1 (since  $e^{-\infty} \rightarrow 0$  and  $e^0 = 1$ ). We rewrite the distance score via the shift-trick to:

$$\hat{d}_m^{[.]}(x_i) := d_m^{[.]}(x_i) - \max_{x_{i'} \in C_m^{[.]}} [d_m^{[.]}(x_{i'})] \quad (7.117)$$

for  $[.] \in \{S, T_j\}$ . We write  $i|k|j$  to indicate that  $t_i \in C_k^{[T_j]}$ , and set  $t_{i|k|j} \equiv t_i$ ; i.e.  $s_i \mapsto i|k|j$  be uniquely identifiable via  $i$ . Then, we define the three (normalized) heuristic scores:

$$\begin{aligned} \text{Sensor distance score: } \varrho_j^{[S]}(s_i) &:= \frac{\hat{d}_j^{[S]}(s_i)}{\sum_{s_{i'} \in C_j^{[S]}} \hat{d}_j^{[S]}(s_{i'})} = \sigma(\hat{d}_j^{[S]}(s_i)) \\ \text{Textual distance score: } \varrho_k^{[T_j]}(t_i) &:= \frac{\hat{d}_k^{[T_j]}(t_i)}{\sum_{t_{i'} \in T_j} \hat{d}_k^{[T_j]}(t_{i'})} = \sigma(\hat{d}_k^{[T_j]}(t_i)) \\ \text{Textual cluster size score: } \varrho_{i|k|j} &:= \sigma \left( |C_k^{[T_j]}| - \max_{C_k^{[T_j]} \in C^{[T_j]}} [|C_m^{[T_j]}|] \right) \end{aligned} \quad (7.118)$$

where  $|.|$  denotes cardinality. The harmonic mean of the clustering scores is:

$$\mathcal{H}(i|k|j) := \frac{3}{\frac{1}{\varrho_j^{[S]}(s_i)} + \frac{1}{\varrho_k^{[T_j]}(t_i)} + \frac{1}{\varrho_{i|k|j}}} = \frac{3\varrho_j^{[S]}(s_i)\varrho_k^{[T_j]}(t_i)\varrho_{i|k|j}}{\varrho_j^{[S]}(s_i)\varrho_{i|k|j} + \varrho_k^{[T_j]}(t_i)\varrho_{i|k|j} + \varrho_j^{[S]}(s_i)\varrho_k^{[T_j]}(t_i)} \quad (7.119)$$

$\mathcal{H}(i|k|j)$  defines a relative ranking of the most appropriate pairs  $(s_{i|k|j}, t_{i|k|j})$ , and thus, we directly obtain a (cluster-wise) ranking for annotations  $t_{i|k|j} \equiv t_i$ . The three heuristics and  $\mathcal{H}(.)$  are depicted in Figure 7.35 below. Although weighting the individual clustering scores is also an option, per default we give each score equal weight and avoid fine-tuning weight parameters.

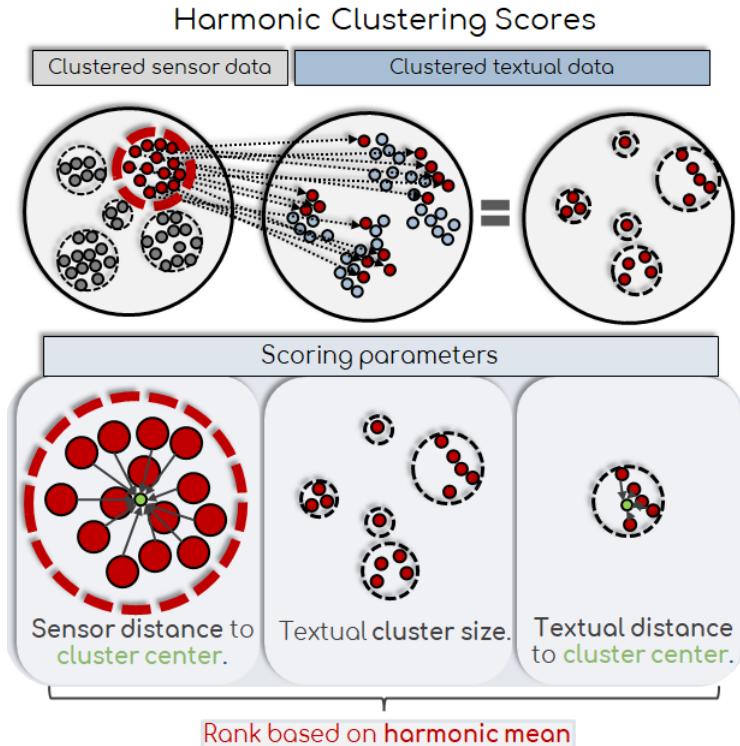


FIGURE 7.35: The Harmonic Clustering Score as the harmonic mean of three intuitive score heuristics. Ranking by this score identifies the most "representative" elements; since achieving a good score requires all three heuristics to yield high scores.

In summary, the Harmonic Clustering Score ranking is determined by three factors: (1) the distance between a sensor data point and the center of its assigned sensor data cluster, (2) the distance between the textual data point and the center of its assigned cluster on the textual data, and (3) the size of the respective projected textual cluster. These three clustering heuristics are combined using the harmonic mean, which penalizes smaller scores strongly, to yield a single scalar representation metric score; providing us so with a score *ranking function*.

Using the ranking obtained from the harmonic mean in this way, we can determine the most appropriate annotation for a given sensor data point. This ranking can also help us identify the most representative sensor data points for a given textual cluster (i.e., the other way round). To do this, we first cluster the textual data and then filter/sub-cluster the sensor data accordingly. Overall, this approach handles large datasets with a high number of sensor data points and textual annotations efficiently and automatically (i.e. it is particularly very scalable to large data volumes); and so, allows us to efficiently identify correlations between different data modalities in large data sets.

### Cluster Centrality Measures for HDBSCAN

Cluster centers are typically represented by either the mean (e.g. center of mass) or the geometric median (the point with the shortest distance to all cluster points). Centroid-based clustering methods also allow the clustering algorithm itself to directly identify the cluster centers (i.e. centroids). However, HDBSCAN is a density-based clustering algorithm that detects clusters by identifying high-density regions separated by low-density regions. Unlike centroid-based techniques like k-means,

HDBSCAN does not assign centers to clusters. Density-based algorithms can work on non-convex geometries, so, computing the average of all points (which may be located outside the cluster) does not necessarily provide an accurate estimate of the cluster's center.

Yet, we can still extract for each individual cluster sample a measure of its *membership strength* to its assigned cluster; especially for HDBSCAN (McInnes, Healy, and Astels, 2017b). Noise points are here assigned a probability of zero, while points within clusters are assigned values that are proportional to their degree of persistence within the cluster. As we apply the Softmax function on the scores in conjunction with the reparametrization shift-trick (Equation 7.117), the final scores for the outliers will not be zero; but rather very small. This is intentional and good, as we do not want to completely discard (potential) outliers during evaluation; notice how if one term in the harmonic mean is zero, the entire harmonic mean is zero (strictly speaking, it is "undefined").

### 7.12.2 Lowest Auto-Regressive Modelling Loss over Cluster Embedding

An alternative approach to identifying the best representative pair  $(s_i, t_i)$  is to rank them based on their auto-regressive modeling loss over their cluster embedding (Figure 7.36). This solution builds on the method explained in Section 7.10, where the aim was to produce a single textual description for a specific cluster. To achieve this, the model was first trained to predict the captions of individual instances, followed by the training of a projection layer (optional) to create a single description for a given cluster over a single corresponding cluster vector representation (representing all instances). Non-representative instances, those with sensor embeddings far from the cluster center or with inadequate or incorrect cluster descriptions, will then have higher (average) convergence loss during prediction and, hence, are expected to be more challenging to train/fit than the representative instances.

- Lowest Auto-Regressive Modelling Loss on Cluster Embedding

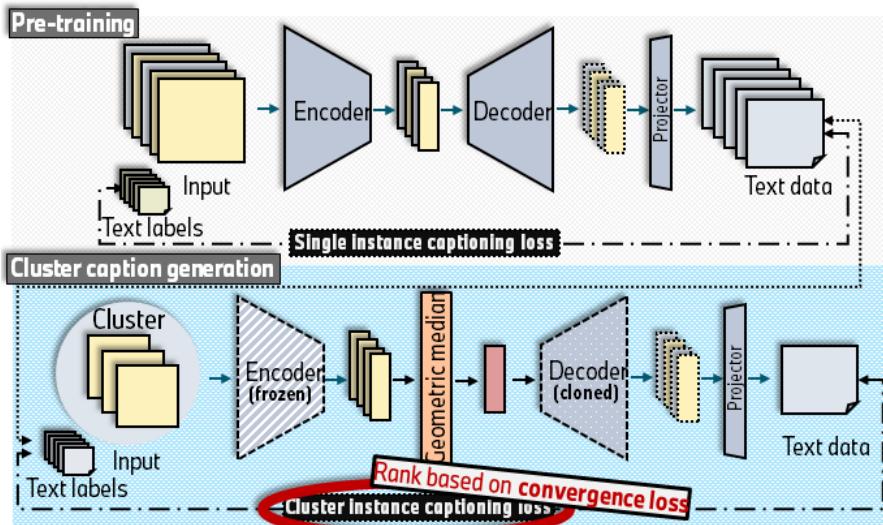


FIGURE 7.36: Identifying the optimal representative pair within a cluster by sorting pairs according to their auto-regressive modeling loss relative to their cluster's embedding

We can organize the pairs  $(s_i, t_i)$  according to their model loss, where lower loss values indicate better representations. Pairs with lower auto-regressive loss values

are considered superior representations of the cluster, as they indicate a smaller difference between the predicted and actual descriptions. Conversely, pairs with high loss values are deemed not representative.

### Example of a Loss Metric Explanation

Consider a group of data points that represent different types of fruits. A model is trained to predict what each fruit is (via clustering), such as "apple", "banana", "orange", and so on. The model also includes a projection layer and an LM-Head that generates a single text description for the whole cluster group of fruits. However, if a data point representing a "car" is present in this group, it will be considered an outlier. The reason is that the model will have difficulty in accurately predicting its description, leading to a higher loss value. The pair associated with this outlier instance would, thus, also be ranked lower respectively. Similarly, suppose we take a "banana" object that is correctly clustered as a fruit instance, but its description is incorrect, such as "This object represents a vehicle". If we pass the "banana" through the trained model, the loss value will also be higher; hence, ranked lower. The reason is that the description is not representative of the group/cluster of fruits and, so, diverges from all the other instances in the cluster.

#### Take-Away

We can evaluate alignments of textual annotations based on their ability to represent clusters discovered in sensor data by determining the harmonic mean of three basic heuristics: sensor cluster centrality, textual cluster size, and textual cluster centrality.

#### Keywords

Uncertain Textual Annotations • Accurate Sensor Data • Harmonic Mean

### 7.13 Transferring Knowledge across Input Domains: Distilling Insights from Machine Learning Models Trained on Different Datasets

Transferring knowledge across different input domains refers to extracting and utilizing insights from ML models that have been trained on different datasets. This might be done due to a variety of reasons, such as reusing knowledge of past data that is no longer accessible; improving model performance; effectiveness, and robustness by leveraging insights from different fields; or generating new explainable rules on more complex data while learning from the labels of easier datasets.

Distilling insights involves methods such as transfer learning, domain adaptation, feature engineering, domain transformation, and model distillation. In our case, we are particularly interested in transferring labeled knowledge from a weaker model to a stronger and larger model. This stronger model has access to a more complex and accurate database that is completely different from the one accessed by the weaker model. Ultimately, the intention is to then extract a third model, which is weaker, but more explainable and solely depends on the new dataset (not the old one); yet still maintaining and respecting the original labels from the first dataset. This is illustrated in Figure 7.37 below:

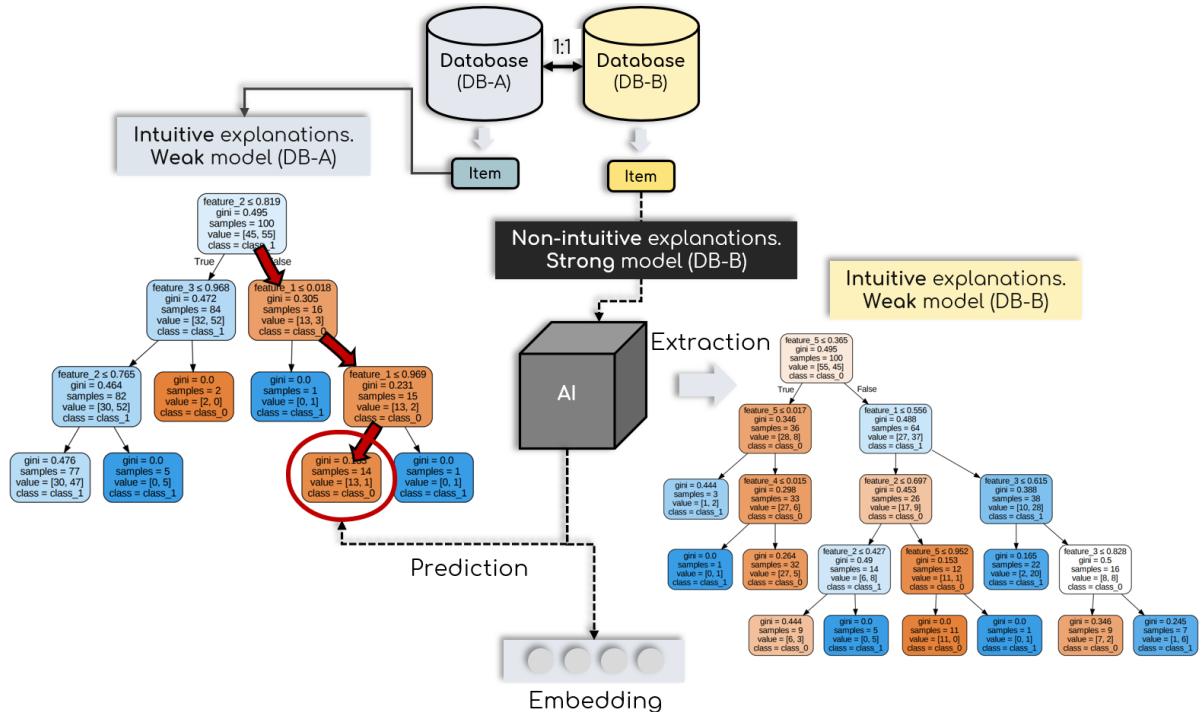


FIGURE 7.37: Transferring knowledge across input domains by distilling insights from models trained/fitted on different data sets.

Alternatively, consider a use case where a model that is not particularly robust, provides labels from a somewhat unreliable and imprecise database. Our goal is to craft a model tailored for a much more precise (but intricate) data source, yet still ultimately aiming for simplicity and ease of explanation; e.g. in label classification. However, simpler models typically sacrifice accuracy, are weaker, and thus much less effective. In particular, creating such a weaker but interpretable model directly from that new, more complex database poses many challenges: e.g., it might easily struggle to extract the original model's knowledge and patterns; or lack the expressiveness needed to identify and extract correlations and non-linear patterns within the more complex data source.

Therefore, a good solution involves implementing and training an *intermediary* ML model. That is, we train a strong and sophisticated robust model (the intermediary model) with the capability to comprehend and extract the (necessary) information inherent in the first dataset by distilling knowledge and prediction labels from the corresponding initial (weaker) model; but instead, and only using the new and complex dataset. Following a successful model training on that complex new dataset, we can leverage this intermediary model to construct a simpler, more explainable new model by distilling interpretable "rules" from it; thus, forming a new *explanatory* model. This now-established interpretable model, therefore, now solely derives its rules directly from the complex dataset; not from the original (e.g. less accurate) dataset. So in essence, this procedure transfers knowledge, adapts it to a new data source, and derives an interpretable model using only explanations over that new data source; thereby rendering old knowledge useful (i.e. reusable) in a completely different and new input domain.

**Take-Away**

Transferring knowledge across input domains involves using insights from ML models trained on different datasets. This can be done through transfer learning, domain adaptation, feature engineering, domain transformation, and model distillation. We can here transfer knowledge from a weaker to a stronger intermediary model, to then extract a simpler, yet more explainable, model (on the new dataset). This repurposes old knowledge from old domains to new domains.

**Keywords**

Knowledge Transfer and Distillation • Explainability and Interpretability • Diverse Data Sources • Domain Adaptation

## 7.14 Model Output Calibration & Novelty Detection

**Please note:** This contribution has been accepted for publication after peer review in Guimerà Cuevas and Schmid, 2024a and has been presented in the conference proceedings.

The objective is to calibrate the output probability estimates of a model for a multi-class classification problem with  $k$  classes, such that the calibrated probabilities reflect the model's prediction confidence both statistically and on a per-instance basis. Notice, how conventional calibration focuses on statistical "on-average" calibration. Given an instance  $i$  in a dataset, let  $\vec{p}_i \in \mathbb{R}^k$  denote the learned output probability vector of the model, where  $\sum_{j=0}^{k-1} p_i^{[j]} = 1$ . So for each class index  $j$ , we define the corresponding one-vs-rest binary classification probabilities as  $p_i^{[j]}$ , representing the probability of instance  $i$  belonging to class  $j$ , and  $\neg p_i^{[j]} = 1 - p_i^{[j]}$ , representing the complement probability of instance  $i$  not belonging to class  $j$ . Note that  $\neg p_i^{[j]} = \sum_{j' \neq j} p_i^{[j']}$ , implying that  $\neg p_i^{[j]}$  is equal to the sum of probabilities of instance  $i$  belonging to all other classes except class  $j$ . This observation will be used later to quantify the level of novelty in the discrepancy between the calibrated probabilities and the original probabilities. Hence, we consider two vectors:  $\vec{p}_i$  and  $\neg \vec{p}_i \triangleq \mathbf{1} - \vec{p}_i$ , where  $\mathbf{1}$  is a vector of ones of the corresponding size.

To calibrate the model, we fit  $k$  separate *class-weighted* one-vs-rest calibration models, denoted as  $\Xi_j$ , where each calibration model is responsible for calibrating the probability estimates of the original model for a specific class  $j$ . Class weighting is essential to address the inherent imbalance in binary one-vs-rest models. Calibration is performed on a distinct development set that does not overlap with the training set used for the original model, ensuring that the calibration models are *not* biased towards the training set.  $\Xi_j$  calibrates the probability estimate of the original model for class  $j$ , ensuring it accurately reflects the true probability of an instance belonging to class  $j$  or not. In essence, it calibrates the probability estimate to make decisions about whether an instance belongs to class  $j$  or not. Since each calibration model  $\Xi_j$  is trained independently, they result in different calibration adjustments for each class. While the goal is usually to maintain class-prediction rankings during calibration, we are particularly interested in examining these differences in calibration among the classes. To capture this, we construct the weight

vector  $\vec{w}_i := [\Xi_0(p_i^{[0]}), \Xi_1(p_i^{[1]}), \dots, \Xi_{k-1}(p_i^{[k-1]})]^T$ , representing the calibrated positive scores for all classes. As it does not form a probability vector, we normalize it to  $\vec{p}_i^* = \vec{w}_i / (\mathbb{1}^T \vec{w}_i)$ . The final calibrated class prediction  $\hat{y}$  is then simply the class index with the highest probability. And so, the calibrated class vector is given by:

$$\vec{p}_i^* = \frac{1}{\sum_{j=0}^{k-1} \Xi_j(p_i^{[j]})} \begin{bmatrix} \Xi_0(p_i^{[0]}) \\ \Xi_1(p_i^{[1]}) \\ \vdots \\ \Xi_{k-1}(p_i^{[k-1]}) \end{bmatrix} \quad (7.120)$$

Let's now consider the following difference  $\epsilon_i^{[\hat{y}]} := p_i^{*[\hat{y}]} - \Xi_{\hat{y}}(p_i^{[\hat{y}]})$ , which is the difference in the calibrated binary class probability over the transformed probability across all other calibrated predictions:

$$\epsilon_i^{[\hat{y}]} = \frac{w_i^{[\hat{y}]} - w_i^{[\hat{y}]} \Xi_{\hat{y}}(p_i^{[\hat{y}]})}{\mathbb{1}^T \vec{w}_i} = \frac{w_i^{[\hat{y}]} (1 - \mathbb{1}^T \vec{w}_i)}{\mathbb{1}^T \vec{w}_i} \quad (7.121)$$

and following, with  $\vec{\epsilon}_i := p_i^* - \vec{w}_i$ , then:

$$\begin{aligned} \mathbb{1}^T \vec{\epsilon}_i &= \sum_{j=0}^{k-1} \epsilon_i^{[j]} = \sum_{j=0}^{k-1} \frac{w_i^{[j]} (1 - \mathbb{1}^T \vec{w}_i)}{\mathbb{1}^T \vec{w}_i} = \frac{(1 - \mathbb{1}^T \vec{w}_i)}{\mathbb{1}^T \vec{w}_i} \sum_{j=0}^{k-1} \vec{w}_i^{[j]} \\ &= \frac{(1 - \mathbb{1}^T \vec{w}_i)}{\mathbb{1}^T \vec{w}_i} \mathbb{1}^T \vec{w}_i = 1 - \mathbb{1}^T \vec{w}_i \end{aligned} \quad (7.122)$$

Knowing that  $\mathbb{1}^T \vec{p}_i \triangleq 1$ , it immediately follows that in a perfectly calibrated model  $\vec{p}_i = \vec{w}_i \implies \mathbb{1}^T \vec{\epsilon}_i = 0$ ; however, the converse is *not* true, i.e.  $\mathbb{1}^T \vec{\epsilon}_i = 0 \nRightarrow \vec{p}_i = \vec{w}_i$ , since  $1 - \mathbb{1}^T \vec{w}_i = 0 \iff 1 = \mathbb{1}^T \vec{w}_i \iff \mathbb{1}^T \vec{w}_i = \mathbb{1}^T \vec{p}_i \iff \vec{w}_i = \vec{p}_i$ .

Knowing that  $\forall j : 0 \leq w_i^{[j]} \leq 1$ , an alternative is to calculate the sum of absolute error differences (the  $L_1$  vector norm), which also eliminates the possibility of positive and negative  $\epsilon_i^{[j]}$  values canceling each other out:

$$\|\vec{\epsilon}_i\|_1 = \mathbb{1}^T |\vec{\epsilon}_i| \triangleq \sum_{j=0}^{k-1} |\epsilon_i^{[j]}| = \sum_{j=0}^{k-1} \left| \frac{w_i^{[j]} (1 - \mathbb{1}^T \vec{w}_i)}{\mathbb{1}^T \vec{w}_i} \right| = \frac{|1 - \mathbb{1}^T \vec{w}_i|}{\mathbb{1}^T \vec{w}_i} \sum_{j=0}^{k-1} \vec{w}_i^{[j]} = |1 - \mathbb{1}^T \vec{w}_i| \quad (7.123)$$

We may also consider the  $L_2$  norm (i.e. Euclidean distance)  $\|\vec{\epsilon}_i\|_2$  instead:

$$\begin{aligned} \|\vec{\epsilon}_i\|_2 &= \sqrt{\sum_{j=0}^{k-1} (\epsilon_i^{[j]})^2} = \sqrt{\sum_{j=0}^{k-1} \left( \frac{w_i^{[j]} (1 - \mathbb{1}^T \vec{w}_i)}{\mathbb{1}^T \vec{w}_i} \right)^2} \\ &= \sqrt{\left( \frac{1 - \mathbb{1}^T \vec{w}_i}{\mathbb{1}^T \vec{w}_i} \right)^2} \sqrt{\sum_{j=0}^{k-1} (\vec{w}_i^{[j]})^2} = \frac{|1 - \mathbb{1}^T \vec{w}_i|}{\mathbb{1}^T \vec{w}_i} \|\vec{w}_i\|_2 \\ &= \|\vec{\epsilon}_i\|_1 \frac{\|\vec{w}_i\|_2}{\mathbb{1}^T \vec{w}_i} = \underbrace{\|\vec{\epsilon}_i\|_1}_{\substack{\text{novelty} \\ \text{confidence}}} \underbrace{\frac{\|\vec{w}_i\|_2}{\|\vec{w}_i\|_1}}_{\text{confidence}} \end{aligned} \quad (7.124)$$

Formula 7.124 breaks down the magnitude of the error into two components. The first component is the magnitude of the difference between 1 and the sum of the calibrated probabilities,  $\|\vec{\epsilon}_i\|_1 = |1 - \mathbb{1}^T \vec{w}_i|$ , and measures how well the independently

calibrated outputs align to a probability distribution. Given a perfect alignment, the sum of the calibrated probabilities is 1, and the component will yield 0. The second component,  $\frac{\|\vec{w}_i\|_2}{\|\vec{w}_i\|_1}$ , represents how "spread out" the elements of  $\vec{w}_i$  are. In general, a vector with a higher quotient (magnitude divided by the sum of its components) will have more "spread out" components, while a vector with a lower quotient will have more concentrated components. Increased variance in the probability predictions indicates that certain predictions carry more probability weight, which in turn implies higher confidence in the model. Conversely, in the absence of variance, such as when output probabilities are evenly or equally predicted, there is a lack of confidence as all outputs have comparable probabilities. The spread is minimal, when all predictions are equal, and maximal when all predictions are zero except one (with then  $p_i^{[y]} = 1$ ); due to the square term dominating the sum term. Consequently, if all components have the same prediction confidence  $w$ , i.e.  $\forall 0 \leq j < k w_i^{[j]} \triangleq w$ , then we have  $\|\vec{w}_i\|_2 = \sqrt{\sum_{j=0}^{k-1} (w_i^{[j]})^2} = \sqrt{kw_i^2} = \sqrt{kw_i}$ . Also,  $\mathbf{1}^T \vec{w}_i = \sum_{j=0}^{k-1} w_i^{[j]} = \sum_{j=0}^{k-1} w_i = kw$ . Therefore, we get  $\frac{\|\vec{w}_i\|_2}{\|\vec{w}_i\|_1} = \frac{w\sqrt{k}}{kw} = \frac{1}{\sqrt{k}}$ . If all classes except one have a zero prediction and the remaining one a prediction of one, then we have  $\|\vec{w}_i\|_2 = 1$  and  $\|\vec{w}_i\|_1 = 1$ . Therefore,  $\frac{\|\vec{w}_i\|_2}{\|\vec{w}_i\|_1} = 1$ . Hence, we have a boundary of  $\frac{\|\vec{w}_i\|_2}{\|\vec{w}_i\|_1} \in [\frac{1}{\sqrt{k}}, 1]$ . On the other hand, the novelty score  $\|\vec{w}_i\|_1$  is in the range of  $[0, k]$ , since  $\epsilon_i^{[j]} \in [0, 1]$ . Therefore,  $\|\vec{\epsilon}_i\|_2 \in [0 \cdot \frac{1}{\sqrt{k}}, 1 \cdot 1] = [0, 1]$  is automatically guaranteed, i.e.  $\|\vec{\epsilon}_i\|_2$  is bounded between zero and one; and increases proportionally with both the level of novelty and confidence.

Novelty measures the degree of unexpectedness, with higher values indicating greater novelty. Confidence represents the level of certainty or assurance, with higher values indicating greater model certainty. Multiplying these two components yields a measure of the magnitude or impact of an (unexpected) prediction. In particular,  $\epsilon_i$  is an instance-specific measure of calibration quality for a class. A positive  $\epsilon_i^{[j]}$  means the model is overconfident in predicting class  $j$ , and the corresponding calibration model  $\Xi_j$  requires adjustments. On the other hand, a negative  $\epsilon_i^{[j]}$  implies that the model is underconfident in predicting class  $j$ , and the calibration adjustment made by the corresponding calibration model  $\Xi_j$  is too strong. If  $\epsilon_i^{[j]}$  is near zero, the model is well-calibrated for class  $j$ . The value of  $\epsilon_i$  reflects the extent of miscalibration, with higher magnitudes indicating greater miscalibration, hence  $\epsilon_i$  can be perceived as a degree of "novelty". The average value of  $\epsilon_i$  offers insight into whether the model is expected to be over or underconfident across all classes. However, its vector magnitude  $\|\vec{\epsilon}_i\|_2$  represents the confidence-weighted novelty, which gives a better understanding of prediction errors by considering both novelty and confidence in the prediction. This helps identify areas where unexpected errors occur with high confidence.

Equation 7.124 also holds true for any arbitrary  $L_\rho$  norm  $\|\vec{\epsilon}_i\|_\rho := \left( \sum_{j=1}^{k-1} |\epsilon_i^{[j]}|^p \right)^{1/\rho}$ ; which can be seen given that:  $\|\vec{\epsilon}_i\|_\rho = \left( \left| \frac{1 - \mathbf{1}^T \vec{w}_i}{\|\vec{w}_i\|_1} \right|^p \right)^{1/\rho} \|\vec{w}_i\|_\rho = \|\vec{\epsilon}_i\|_1 \frac{\|\vec{w}_i\|_\rho}{\|\vec{w}_i\|_1}$ . Notice, how in particular, the  $L_\infty$  norm corresponds per definition to the maximum value in a vector, and therefore:

$$\|\vec{\epsilon}_i\|_\infty = \|\vec{\epsilon}_i\|_1 \frac{\|\vec{w}_i\|_\infty}{\|\vec{w}_i\|_1} = \|\vec{\epsilon}_i\|_1 \frac{\bar{w}_i^{[y]}}{\|\vec{w}_i\|_1} = \underbrace{\|\vec{\epsilon}_i\|_1}_{\text{novelty}} \cdot \underbrace{p_i^{*[y]}}_{\text{max. confidence}} \quad (7.125)$$

Thus, when considering the  $L_\infty$  norm, the second term is the maximum confidence value (i.e. output probability of its prediction  $\hat{y}$ ).

### 7.14.1 Score Adjustment

To calculate the error norm  $\|\vec{e}\|$ , we need to consider the impact of calibration mismatches on low probability class scores compared to high scores. Larger scores have a diminishing effect on smaller scores in the probability distribution. For example, if the predicted class has no mismatch but other classes do, the contribution from those mismatches will be limited if the main class has a high probability. Or to put it in a different way, consider a scenario where there are no mismatches in the predicted class (the class with the highest Softmax score), but there are mismatches among the other classes. If the main class has a high probability (e.g. 95%), any additional contribution from the other mismatches will be considerably reduced and limited within the remaining probability mass (i.e. 5%).

To address this issue, one approach is to assign weights to each mismatch score based on their (normalized) complement probability:

$$\|\vec{e}_i\|_\rho^\infty := \left\| \frac{k(\mathbb{1} - \vec{p}_i^*)}{\mathbb{1}^T(\mathbb{1} - \vec{p}_i^*)} \circ \vec{e}_i \right\|_\rho = \left\| \frac{k(\mathbb{1} - \vec{p}_i^*)}{k-1} \circ \vec{e}_i \right\|_\rho \triangleq \|\vec{\alpha}_i \circ \vec{e}_i\|_\rho \quad (7.126)$$

where  $\circ$  is the Hadamard product and  $\vec{\alpha}_i$  be the proportional normalized complement weight vector; multiplying by the class size  $k$  is a common practice when working with sample weights, as it ensures that the weighted average of the samples is equal to the unweighted average. Yet, given a closer look, we can realize that  $\|\vec{e}_i\|_\rho^\infty$  is proportional to the following:

$$\|\vec{e}_i\|_\rho^\infty \triangleq \|\vec{\alpha}_i \circ \vec{e}_i\|_\rho = \frac{k}{k-1} \|\neg \vec{p}_i^* \circ \vec{e}_i\|_\rho \sim \|\neg \vec{p}_i^* \circ \vec{e}_i\|_\rho \quad (7.127)$$

where  $k$  is the number of classes. Considering mismatches for low probability classes leverages “dark knowledge” Hinton, Vinyals, and Dean, 2015, i.e. knowledge encoded in the relative probabilities of incorrect outputs, which we can use for mismatch scoring.

### 7.14.2 Calibration Discrepancy with Label Aggregations

Let  $\mathcal{C}$  be a set of  $k$  distinct classes. Our goal is to quantify the calibration error discrepancies for each class. To capture additional error differences, we suggest extending the error vector  $\vec{e}$  to higher dimensions by combining pairwise labels using a subset or all  $\binom{k}{2}$  label pairs under a single label and evaluating their calibration accuracy. Let us denote an arbitrary instance as  $i$  and its corresponding probability vector as  $\vec{p}_i$ , where  $p_i^{[j]}$  represents the probability assigned to class  $j$ . Given that the prediction is determined by the class index with the highest probability (i.e. the argmax operation), the probabilities  $p_i^{[j]}$  and  $p_i^{[j']}$  are mutually exclusive for  $j \neq j'$ ; meaning that their joint-conditional probability is zero for any pair of classes, i.e.  $P(\hat{y} = j \cap \hat{y} = j') = 0 \implies P(\hat{y} = j | \hat{y} = j') = 0$ . To ensure clarity in notation, we introduce  $P_i(j)$  to denote the probability of the predicted class being  $j$  for instance  $i$ . With this notation, we can express the probability  $P_i(j \cup j') \triangleq P_i(j) + P_i(j') - P_i(j \cap j') \equiv P_i(j) + P_i(j')$ ; for  $j \neq j'$ . Rewriting this expression gives  $0 = P_i(j \cup j') - P_i(j) - P_i(j')$ . Since we can identify each class combination

pair  $(j, j')$  by an enumeration, without loss of generality  $j < j'$ , we define an indexing function  $\chi(j, j') \mapsto \{1, \dots, k\}$  to uniquely identify each pair by an index. Hence, we define the difference:

$$\epsilon_i^{\bullet[\chi(j, j')]} = \Theta[P_i(j \cup j')] - \Theta[P_i(j)] - \Theta[P_i(j')] \quad (7.128)$$

where  $\Theta[\cdot]$  is the calibration function. Ideally,  $\epsilon_i^{\bullet[\chi(j, j')]} = 0$ , indicating that the aggregated pairwise labels do not exhibit any calibration discrepancy. Similar to Equation 7.127, we can incorporate the normalized complement weight vector  $\vec{\alpha}_i^{\bullet}$  with components  $\vec{\alpha}_i^{\bullet[\chi(j, j')]} := \frac{\binom{k}{j}(1-\Theta[P_i(j \cup j')])}{\sum_{\kappa \neq \kappa'} 1-\Theta[P_i(\kappa \cup \kappa')]}.$

Analogously, let  $\lambda$  be an integer such that  $1 < \lambda < k$ . We can generalize the formula for multiple different sized  $\lambda$ -tuple class pairs  $J_\lambda$  of size  $\lambda$  as:

$$\epsilon_{i/\lambda}^{\bullet[\chi(J_\lambda)]} = \Theta[P_i(\bigcup_{j \in J_\lambda} j)] - \sum_{j \in J_\lambda} \Theta[P_i(j)] \quad (7.129)$$

The normalization in  $\vec{\alpha}_{i/\lambda}^{\bullet}$  occurs analogously; with  $\mathbb{1}^T(\mathbb{1} - \vec{p}_n) = \binom{k}{n} - \binom{k-1}{n-1}$  (Proof A.1). The total loss is then the magnitude of the concatenated class error discrepancies and all  $\lambda_n$ -pair aggregations for each instance:

$$\|\vec{\epsilon}_i\|_\rho^\infty = \left\| \underbrace{(\vec{\alpha}_i \circ \vec{\epsilon}_i)}_{k \text{ class errors}} \stackrel{H}{\oplus} \underbrace{(\vec{\alpha}_{i/\lambda_1}^{\bullet} \circ \vec{\epsilon}_{i/\lambda_1}^{\bullet})}_{\binom{k}{\lambda_1} \text{ class-pair errors}} \stackrel{H}{\oplus} \dots \stackrel{H}{\oplus} \underbrace{(\vec{\alpha}_{i/\lambda_n}^{\bullet} \circ \vec{\epsilon}_{i/\lambda_n}^{\bullet})}_{\binom{k}{\lambda_n} \text{ class-pair errors}} \right\|_\rho \quad (7.130)$$

where  $\stackrel{H}{\oplus}$  represents the horizontal-concatenation operation. Since the number of pairs scales binomially, but we have the identity  $\binom{k}{\lambda} = \binom{k}{k-\lambda}$ , one simple choice for the values of  $\lambda$  would be to use  $\lambda_1 := 2, \lambda_2 := k-2$ . Alternatively, we could choose any two distinct values for  $\lambda_1$  and  $\lambda_2$  such that  $1 < \lambda_1 < \lambda_2 < k$ . However, choosing larger values for  $\lambda$  increases the number of pairs considerably due to the rapid growth of the binomial coefficient.

### 7.14.3 Hybrid OOD: Embedding- and Prediction-based

We can make OOD decisions by combining the results of multiple models, such as averaging or selecting the maximum value. This allows us to create a hybrid OOD outlier score function for a new instance  $z_i$ , using the maximum of the base model and the calibration mismatch over their unified probability scores. Unified scores (Kriegel et al., 2011) are necessary because raw scores from different models may not be directly comparable. We define  $\Phi_j^{\mathcal{X}}$  as the unifying score function for a model  $\Pi_j$  over instances in  $\mathcal{X}$ . The hybrid outlier score  $\pi$  is then given by:

$$\pi(z_i) = \max\{\Phi_0^{\mathcal{X}}(\Pi_0(z_i)), \dots, \Phi_j^{\mathcal{X}}(\Pi_j(z_i)), \Phi_{j+1}^{\mathcal{X}}(\|\vec{\epsilon}_i\|_\rho^\infty)\} \quad (7.131)$$

where  $\Phi_j^{\mathcal{X}}$  is the "unifying score function" for  $\Pi_j$  over  $\mathcal{X}$  (Kriegel et al., 2011):

$$\begin{aligned} \text{erf}(x) &:= \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \\ \Phi_j^{\mathcal{X}}(z_i) &:= \frac{1}{2} \left[ 1 + \text{erf} \left( \frac{\Pi_j(z_i) - \mu_{\mathcal{X}}}{\sigma_{\mathcal{X}} \sqrt{2}} \right) \right] \end{aligned} \quad (7.132)$$

$\mu_{\mathcal{X}}, \sigma_{\mathcal{X}}$  are respectively the mean and standard deviation of  $\Pi_j(\mathcal{X})$ ; The Gauss error,  $\text{erf}(x)$ , represents the likelihood that a single measurement's error  $x > 0$  falls within a fixed range. It is known that, for a normal distribution with zero mean and standard deviation of  $\frac{1}{\sqrt{2}}$ ,  $\text{erf}(x)$  gives the probability of the variable being within the symmetric range of  $[-x, x]$ . However, if the variable conforms to a normal distribution with a standard deviation  $\sigma$  (and mean zero), the probability is given by  $\text{erf}(\frac{x\sqrt{2}}{\sigma})$ . Overall, by dividing the result by 2 and adding 1, we ensure that the probability falls between 0 and 1.

## 7.15 Percentile Ranked Scores

To handle uneven distributions of raw probability scores, we suggest calibrating scores based on their percentiles relative to other scores. Outliers will here deviate more likely from the reference data distribution, making it easier to detect anomalies. When calibrating probabilities, an outlier's class probability may draw mass from other predictions, leading to significant percentile variations due to ranking changes. We use the formula  $\Xi_j^{\Psi}(p_i^{[j]}) := \Xi_j(\Psi(p_i^{[j]}))$ , where  $\Psi(\cdot)$  is the percentile transformation that converts probability data into percentiles using reference data; the other formulas are updated accordingly. The percentile transformation finds the left and right indices of the input probabilities in sorted reference data and calculates the percentiles as their average.

### Take-Away

Model-generated probabilities can be calibrated to improve confidence estimation and accuracy. In a multi-class setup, using separate calibration models for each class allows comparison between predicted and actual values. This helps identify outliers that do not fit expected training data distributions; showing poorer calibration alignment precisely due to being out-of-distribution (since the calibration itself was trained on in-distribution data). Error metrics norms like  $L_1$  and  $L_2$  quantify such misalignments, revealing different insights into model miscalibration and unexpected prediction errors.

### Keywords

Model Calibration • Out-of-Distribution Detection • Calibration Discrepancy

## 7.16 Representative Data Stream Sampling for Trajectories

Let  $T_N$  be a spatial-temporal trajectory in a multi-dimensional vector space, represented as  $T_N = ((\vec{z}_1, t_1), (\vec{z}_2, t_2), \dots, (\vec{z}_N, t_N))$ , where  $N$  denotes the number of data points in the current trajectory. In this context, we define a *quantile-subset*  $Q_{\phi} \subset T$  as:

$$Q_{\phi} := \{(\vec{z}_i, t_i) \in T \mid i \geq \arg \min_{j < N} |j - \phi \cdot N|\} \quad (7.133)$$

Essentially,  $Q_{\phi}$  is a subset of a trajectory that starts from the  $\phi$ -th quantile position and continues to the end. This is useful for focusing on a specific subset of the trajectory for analysis or processing. When  $\phi$  approaches 1, it focuses on the latter

(i.e. the newer) part of the trajectory, hence, including data points closer to the end. Conversely, when  $\phi$  approaches 0, then respectively on the entire (i.e. full) trajectory.

### 7.16.1 Decaying Quantile Chain-Sampling

We consider the challenge of maintaining  $Q_\phi$  given a continuous data stream trajectory with an unknown or infinite size. We can only store a limited number of samples, denoted as  $k$ ; thus  $|Q_\phi| \leq k$ . We employ an adaptation inspired by Reservoir Chain Sampling, named Decaying Quantile Chain-Sampling (DQCS) (Alg. 2).

---

**Algorithm 2:** Decaying Quantile Reservoir Sampling (DQRS-Update)

---

**Indexing:** Starting at one;

**Input:**  $e_i$  (element at time-step  $i$ ),  $\phi$  (quantile),  $k$  (max. size),  $Q$  (reservoir);

**Notation:**  $\mathbb{U}|Q_\phi^{\{j\}}| := \max(1, |Q_\phi^{\{j\}}|)$ , and  $|Q_\phi^{\{j\}}|$  is the reservoir size at time-step  $j$ ;

**Function** RandomUnusedQuantileIndex( $Q, \phi, a, b$ ):

```

 $\Delta_b^\phi \leftarrow \lceil \phi \cdot b \rceil;$ 
 $\Omega \leftarrow \{c \in \mathbb{N} \setminus Q.\text{KEYS}() \mid a + 1 \leq c \leq a + \Delta_b^\phi\};$ 
 $P(e_i) := \sum_{j=\lfloor \frac{i-1}{1+\phi} \rfloor + 1}^{i-1} \mathbb{U}|Q_\phi^{\{j\}}| P(e_j) \lceil \phi j \rceil^{-1};$ 
return  $\begin{cases} \text{RANDOM}(\Omega) \sim \frac{1}{P(e_i)} & \text{if } \Omega \neq \emptyset \\ \text{NONE} & \text{otherwise} \end{cases}$ 
if  $i \in Q.\text{KEYS}()$  then
|  $(e_j, \tau) \leftarrow Q[i]$ ;
| DELETE  $Q[i]$ ;
|  $t \leftarrow \text{RandomUnusedQuantileIndex}(Q, \phi, j, i)$ ;
|  $\Delta_\tau^\phi \leftarrow \lceil \phi \cdot \tau \rceil$ 
| if  $t \neq \text{NONE} \wedge t - j > \Delta_\tau^\phi$  then
| |  $Q[t] \leftarrow (e_j, i)$ ;
| | return;
| end
| else if  $|Q| \geq k$  then
| | return;
| end
|  $t = \text{RandomUnusedQuantileIndex}(Q, \phi, i, i)$ ;
| if  $t \neq \text{NONE}$  then
| |  $Q[t] \leftarrow (e_i, i)$ ;
| end

```

---

The DQCS-Update method employs a sampling approach within a specified quantile window, drawing elements from a set without replacement. It operates using a quantile-reservoir  $Q$  with a maximum capacity of  $k$  as follows: when time-step  $j$  resides in  $Q$ , it links to an index  $t_j$  representing a forthcoming item to replace the current one. This replacement index is selected from the respective quantile window of the current step, inversely proportional to each index's reachability probability in the chain, ensuring a final sample based on a uniform distribution. Upon the arrival of time-step  $t_j$ , a potential new free (non-linked) future chain index  $t$  is sampled within the  $\Delta_\tau^\phi$  quantile window, with  $\tau$  initially set to  $j$ . If a free  $t$  is found and  $t > \Delta_\tau^\phi$ ,  $\tau$  is updated with the new replacement index  $t$ , and the chain-replacement

index of  $e_j$  is reassigned to  $e_t$ . Otherwise, if either the time-step is not a chain index or the reservoir is not full ( $|Q| < k$ ), the current item is replaced (if a chain index) or inserted (if  $|Q| < k$ ) into the reservoir. Afterwards, in either case, a future chain index within the current quantile window range is then, again, randomly chosen.

"Chain"-sampling occurs automatically, as a new item's replacement index is then immediately assigned; forming a "chain" of replacement indices. The size of the replacement window grows based on the quantile  $\phi$ . When a replacement is needed for a reservoir item due to a replacement-chain index at time-step  $i$ , the replacement probability considers the new window size and current quantile ratio. For time-shift adjustment, the algorithm compares the previously sampled future chain index  $t_j$  with a newly sampled index  $t_i$  within the current window size. If  $t_i$  exceeds the previous window size ( $t_i - j > \Delta_\tau^\phi$ ), the replacement chain index updates  $t_i$  as the new chain replacement index while keeping the reservoir item unchanged ( $t_j$  is ignored). Otherwise, the item in the reservoir is replaced by the current item of time-step  $t_j$ . This ensures the newly sampled index always falls within the quantile range to replace expired items in the quantile range while updating the sampling probability to reflect the new (current) quantile window size. This is because chaining consistently selects subsequent indices within the quantile window, and if an item has expired,  $t_i$  cannot surpass  $\Delta_\tau^\phi$ , since the minimum possible new index would be outside the quantile window.

The reasoning behind this is motivated as follows: We use the short-notation  $\mathbb{I}_{a|b} := [a + 1, a + \Delta_b^\phi]$  to denote intervals. Now, consider an item in a reservoir sampled at time-step  $j$  with a chain-replacement time index  $t_j$  within the interval  $\mathbb{I}_{j|j} = [j + 1, j + \Delta_j^\phi]$ , where  $\Delta_j^\phi = \lceil \phi j \rceil$  is the length of the interval, denoting the current quantile window size for  $0 < \phi < 1$ . In standard chain sampling, when the current time-step is  $i = t_j$ , the item is immediately replaced. However, in our scenario, the window size depends on  $i$  as  $\Delta_i^\phi = \lceil \phi t_i \rceil$ . This causes a discrepancy between the probability of selecting the replacement index at time-step  $j$  and that at  $i$ , due to the altered window size. To resolve this, we uniformly and independently sample another index  $t$  within the interval  $\mathbb{I}_{j|i} = [j + 1, j + \Delta_i^\phi]$  and compare it to the old replacement index  $t_j$ . If  $t - j > \Delta_j^\phi$ , it becomes the new replacement index, and the item in the reservoir is not yet replaced. Let's also now further consider the probability of an item at a (current) time-step  $i$  to be selected in the set of chain-mappings  $M^{\{\cdot\}} := Q_\phi^{\{\cdot\}}.\text{KEYS}()$ ;  $M$  denotes the current one. Let  $Q_\phi^{\{i\}}$  denote the reservoir at time-step  $i$ . We know  $|Q_\phi^{\{i\}}| \leq \lceil \phi i \rceil \leq k$ . Every item is chained to a uniform random item within the quantile window size  $\Delta_i^\phi$ . Thus, every item  $i > 1$  itself has  $\Delta_{i-1}^\phi$  many possible incoming paths. This forms a reachability graph (Figure 7.38).

The probability of reaching a particular node by following a single random path is the summation of probabilities that: the previous nodes connected to it are reached times that link leading to the node of interest is randomly uniformly selected from all its outgoing links. According to the law of total probabilities, given  $P(1) := 1$ , this can be (recursively) defined for  $i > 1$  given a set of indices  $\mathcal{R}(i)$  from which  $i$  can be

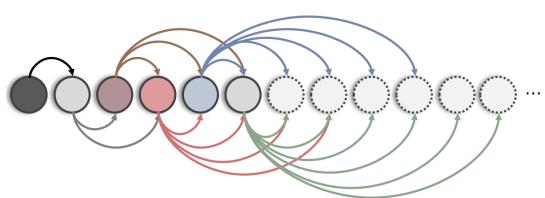


FIGURE 7.38: Sequence chaining for  $\phi = 1$

reached (i.e. chained) for  $0 < \phi \leq 1$ , as:

$$P(i) := \sum_{j \in \mathcal{R}(i)} P(j) \times \frac{1}{\Delta_j^\phi} = \sum_{j=\min\{j \geq i - \lceil \phi j \rceil\}}^{i-1} \frac{P(j)}{\lceil \phi j \rceil} = \sum_{j=\lfloor (\frac{i-1}{1+\phi}) \rfloor + 1}^{i-1} \frac{P(j)}{\lceil \phi j \rceil} \quad (7.134)$$

Notice how:  $\lfloor (\frac{i-1}{1+\phi}) \rfloor + 1 = \min\{j \geq i - \lceil \phi j \rceil\}$  offers a direct analytical solution for retrieving the starting index  $j$ .

In practical applications, precise floating-point accuracy holds importance here, due to the strict truncation of the "floor" operation. Without it, there might be occasional deviations around one index off the minimum value. However, identifying and fixing such minor errors is straightforward; simply by checking if the index satisfies the condition; and correcting otherwise.

As  $j \geq i - \lceil \phi j \rceil \geq i - \phi j - 1$ , we know  $j \geq \lfloor (\frac{i-1}{1+\phi}) \rfloor + 1 \geq \frac{i-1}{1+\phi} \geq \lfloor (\frac{i-1}{1+\phi}) \rfloor$ . Let  $j^* := \min\{j \geq i - \lceil \phi j \rceil\}$ . Then,  $j^* \geq i - \lfloor (\frac{i-1}{1+\phi}) \rfloor$ , must also hold, but this implies  $j^* = \lfloor (\frac{i-1}{1+\phi}) \rfloor$  as this would otherwise contradict that  $j^*$  is the minimum integer satisfying this condition.

With reservoir size  $|Q_\phi|$ , multiple chain-paths exist. Let  ${}^\cup|Q_\phi^{\{t_j\}}| := \max(1, |Q_\phi^{\{t_j\}}|)$  denote a minimum count of one. Notice how  $|Q_\phi^{\{t_j\}}| \leq \lceil \phi j \rceil$ . Subsequently, it is  $P_{Q_\phi}(1) := 1$  and:

$$P_{Q_\phi}(i) := \sum_{j=\lfloor (\frac{i-1}{1+\phi}) \rfloor + 1}^{i-1} \frac{{}^\cup|Q_\phi^{\{j\}}|}{\lceil \phi j \rceil} \times P_{Q_\phi}(j) \quad (7.135)$$

If we know we reached an index  $t < i$  in any mapping  $M^{\{<i\}}$  before, then:

$$P_{Q_\phi}(i \mid t \in M^{\{<i\}}) := \sum_{j=\lfloor (\frac{i-1}{1+\phi}) \rfloor + 1}^{i-1} \frac{{}^\cup|Q_\phi^{\{j\}}|}{\lceil \phi j \rceil} \times \begin{cases} P_{Q_\phi}(j \mid t \in M^{\{<i\}}) & \text{if } j > t \\ 1 & \text{if } j = t \\ 0 & \text{otherwise} \end{cases} \quad (7.136)$$

The probabilities are depicted in Figure 7.39 below; exemplary for the initial chaining-probability and the conditional chaining-probability:

$P_{Q_\phi}(i)$  can also be explicitly expressed in matrix form (instead of recursive). Lets denote  $\lambda_{|i} := \lfloor (\frac{i-1}{1+\phi}) \rfloor + 1$  and  $\rho(j', j) \begin{cases} \frac{1}{\Delta_{j'}^\phi} & \text{if } j > j' \geq \lambda_{|j} \\ 0 & \text{otherwise} \end{cases}$ ; then:

$$P_{Q_\phi}(i) = [\rho(i - \lambda_{|i}, i) \quad \cdots \quad \rho(i - 1, i)] \begin{bmatrix} P_{Q_\phi}(\lambda_{|i}) \\ \vdots \\ P_{Q_\phi}(i - 1) \end{bmatrix} \quad (7.137)$$

And since, without loss of generality assuming the array be sorted in ascending order with  $a$  the smallest and  $z$  the largest index, it is:

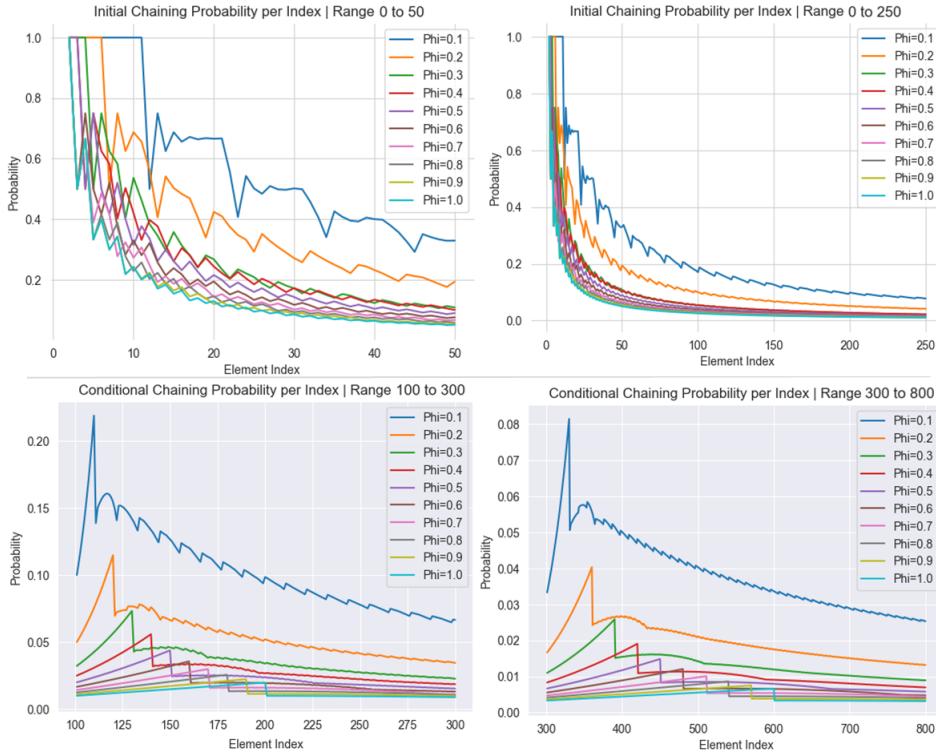


FIGURE 7.39: The single-path forward chaining probability for various quantile values  $\phi$ .

$$\begin{bmatrix} P_{Q_\phi}(a) \\ \vdots \\ P_{Q_\phi}(z) \end{bmatrix} = \begin{bmatrix} \rho(\lambda_{|a}, a) & \cdots & \rho(\lambda_{|z}, a) & \cdots & \rho(z-1, a) \\ \vdots & \ddots & \vdots & \vdots & \ddots \\ \rho(\lambda_{|a}, z) & \cdots & \rho(\lambda_{|z}, z) & \cdots & \rho(z-1, z) \end{bmatrix} \begin{bmatrix} P_{Q_\phi}(\lambda_{|a}) \\ \vdots \\ P_{Q_\phi}(\lambda_{|z}) \\ \vdots \\ P_{Q_\phi}(z-1) \end{bmatrix} \quad (7.138)$$

And so, we can repeatedly stack matrices into one closed-form solution:

$$P_{Q_\phi}(i) = [\dots] \times \left( \left[ \begin{array}{ccc} \dots & & \end{array} \right] \times \left( \dots \times \left( \left[ \begin{array}{ccc} \dots & & \end{array} \right] \times \left[ \begin{array}{c} \vdots \\ \vdots \end{array} \right] \right) \right) \right) \quad (7.139)$$

However, this explicit matrix form needs quadratic matrix storage space; but can now perform optimized vectorized matrix operations. Also, notice how inner matrices are often sparse towards the end.

Size  ${}^\psi|Q_\phi^{\{j\}}|$  is limited to  $k$  and is the number of chains at time-step  $j$ . For a single chaining path, the term  ${}^\psi|Q_\phi^{\{j\}}|$  in  $P_{Q_\phi}(i \mid m \in M^{\{<i\}})$  is replaced with *one*, denoted via  $P(i \mid m \in M^{\{<i\}})$ ; here we only have a single spot in the reservoir. A different perspective here is to consider this as the result of reaching different indices based on the number of paths (rather than the reachability likelihood). Similarly, the number of possible different incoming chain-paths for  $i > 1$  can be defined recursively, given

$\kappa(1) := 0$ , as the number of totally different paths in the previous step plus the number of incoming steps :

$$\kappa(i) := \kappa(i-1) + (i-1) - \lambda = \kappa(i-1) + i - 2 - \lfloor \left( \frac{i-1}{1+\phi} \right) \rfloor \quad (7.140)$$

Similarly, if we know we have already reached index  $t$ , then:

$$\kappa(i \mid t \in M^{\{<i\}}) := i - 2 - \lfloor \left( \frac{i-1}{1+\phi} \right) \rfloor + \begin{cases} \kappa(i-1 \mid t \in M^{\{<i\}}) & \text{if } i > t \\ 0 & \text{otherwise} \end{cases} \quad (7.141)$$

To achieve a uniform distribution, we use a transition probability that is inversely proportional to its reachability likelihood (Figure 7.40):

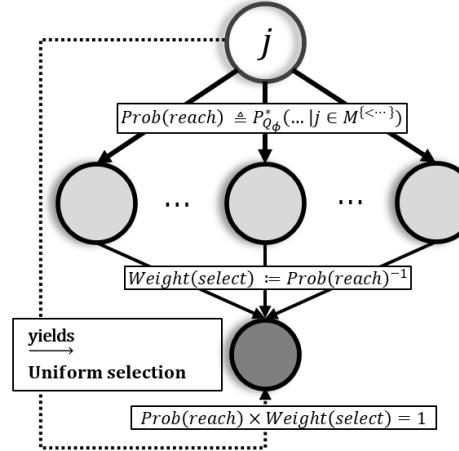


FIGURE 7.40: Inverse weighting relative to reachability likelihood.

However, for that, we need to determine the state transition probabilities of reaching a specific index given a current index; which requires adjusting the transition probability of one node, which in turn impacts the probability of reaching subsequent nodes. This means we must re-adjust the reachability formula, but this creates a recursive cycle that depends on both past and future probabilities and precludes a closed-form solution. Consequently, we must iteratively approximate the probability, beginning with  $P_{[1]}^*(i) := P(i)$ , and iterating for  $m > 1$ .

$$P_{[m]}^*(i) \leftarrow \sum_{j=\lambda}^{i-1} \frac{P_{[m-1]}^*(i)^{-1}}{\sum_{j'=j+1}^{j+\Delta_j^\phi} P_{[m-1]}^*(j')^{-1}} \times P_{[m]}^*(j) = P_{[m-1]}^*(i)^{-1} \sum_{j=\lambda}^{i-1} \frac{P_{[m]}^*(j)}{\sum_{j'=j+1}^{j+\Delta_j^\phi} P_{[m-1]}^*(j')^{-1}} \quad (7.142)$$

where,  $m$  represents the number of iterations. It be  $P^* := P_{[\infty]}^*$ . The conditional probability formula  $P_{[m]}^*(i \mid t \in M^{\{<i\}})$  updates in a similar way:

$$P_{[m]}^*(i \mid t \in M^{\{<i\}}) \leftarrow \sum_{j=\lambda}^{i-1} \frac{P_{[m-1]}^*(i \mid t \in M^{\{<j+1\}})^{-1}}{\sum_{j'=j+1}^{j+\Delta_j^\phi} P_{[m-1]}^*(j' \mid t \in M^{\{<j+1\}})^{-1}} \times P_{[m]}^*(j \mid t \in M^{\{<i\}}) \quad (7.143)$$

Yet, this approximation quickly becomes too expensive for large iterations due to the cyclic nature of future and past probabilities. As a solution, a few observations

can be computed and curve fitting prediction can be utilized to impute missing data for all indices as an approximation like in regression. However, this technique is primarily important mostly for early indices where probability reachability differences are (still) significant. After a certain number of indices, the differences become more and more *insignificant* and can eventually be neglected. Thus, a truly perfectly uniform sample is rather more useful for theoretical analysis than for practical application. Therefore, alternatively, a somewhat less precise but much more efficient approximation can be used as a trade-off; given an index threshold  $\theta \in \mathbb{N}^+$  (analogously for the conditional probability):

$$P_{[m]}^*(i) \stackrel{\Delta}{\approx} P_{[\theta]}^*(\min[i, \theta])^{-1} \sum_{j=\wedge}^{i-1} \frac{P_{[m]}^*(\min[j, \theta])}{\sum_{j'=j+1}^{j+\Delta_j^\phi} P_{[\theta]}^*(\min[j', \theta])^{-1}} \quad (7.144)$$

Given such a threshold, the probability vector can be determined more efficiently using a basic linear algebra trick. This involves employing a transition matrix, which encodes the probability of transitioning from one index to another. The  $\theta$ -*explicit* reachability probability state matrix, denoted as  $S_{[m]}$ , stores the transition probabilities for the first  $\theta$  indices. The term  $\theta$ -*explicit* indicates that the matrix is **explicitly** provided for indices  $j$  and  $i$ , where  $j, i \leq \theta$ , but **implicitly** otherwise; i.e.,  $S_{[m]}^{[j,i]} \leftarrow S_{[m]}^{[\min(\theta,j), \min(\theta,i)]}$ . This saves both computation and memory. Initially, the  $j$ -th row is set to be that of the reachability probability state, i.e.  $S_{[1]}^{[j,:]} := P(i)$ . Then, the state of the  $m$ -th approximation can then be determined via the transition matrix  $T$  up to a time-step  $t \geq i, j$ :

$$\begin{aligned} T_{[m]}^{[j,i]} &= S_{[m]}^{[j,i]}^{-1} \underbrace{\sum_{c=j+1}^{j+\Delta_j^\phi} S_{[m]}^{[j,c]}^{-1}}_{\text{row-sum.}} \\ S_{[m+1]}^{[j,i]} &= \sum_{0 < c \leq t} \underbrace{S_{[m]}^{[j,c]} T_{[m]}^{[c,i]}}_{\text{matrix-mult.}} \end{aligned} \quad (7.145)$$

In cases where there is a division by zero, the respective entry is set to zero. This faster computation, combined with pre-computed probability approximations or caching past results, results in an efficient approximation. The conditional approximation follows a similar approach but has the advantage of using a dynamic threshold. Again, this can be achieved by setting the respective entries in the transition matrix to zero; "removing" past or expired instances. That is, old entries can be omitted. The matrix can also be dynamically reshaped by setting e.g.  $\theta' := \theta + t$ , where  $t$  represents the most recent observation of interest. Similarly, to establish a lower bound threshold  $\theta$ , setting  $\theta' := \min(\theta, t)$ . When using multiple chaining paths,  $t$  should be the minimum of all the observed paths.

Overall, by traversing a path using the (approximated) inverse probability, we can approximate a uniform distribution. In other words, if well approximated, the graph traversal provides an about uniform selection with  $\frac{1}{|\phi|}$  probability on the final transition probabilities for choosing the next chain on a single path; i.e., we multiply the transition probabilities by the inverse probability  $P^*(t_j \mid j \in M^{\{<t_j\}})^{-1}$ .

A uniform probability means that each index has an equal chance of being chained. Overall, given  ${}^{\wedge}|\phi|$  many potential replacement spots in the reservoir at time-step  $i$ , there are equally many (non-independent) concurrent chaining paths. So, the

probability of chaining  $\Pr\{t_j \in M\}$  is:

$$\Pr\{t_j \in M\} \triangleq \frac{\mathbb{E}|Q_\phi^{\{t_j\}}|}{\lceil \phi j \rceil} \quad (7.146)$$

The chaining probability fluctuates greatly at the start of a stream but then decreases rapidly as the stream size increases. Already after some time-steps, this discrepancy becomes quickly negligible. To avoid computational overhead, we can use an approximate error threshold  $\epsilon$  and assume the reachability probability is  $1/\lceil \phi i \rceil$  once the discrepancy is below this threshold. In that case, we have the approximation  $\Pr\{t_j \in M \mid \epsilon\} \approx \frac{k}{\lceil \phi j \rceil}$  (by that time the reservoir is assumed to be full; i.e. no empty spots).

At a specific time-step  $j$ , the likelihood of chain-selecting item  $t_j$  and subsequently adding it to the reservoir at time-step  $i = t_j$ , thus not re-chaining  $j$  to a new future random free time-step index  $t$ , is expressed as follows:

$$\Pr\{t_j \in M \wedge t - j \leq \Delta_j^\phi\} = \Pr\{t_j \in M\} \times \Pr\{t_j - j \leq \Delta_j^\phi \mid t_j \in M\} \quad (7.147)$$

where  $t_j \sim \mathbb{I}_{j|j}$ ,  $t \sim \mathbb{I}_{j|i}$  are inversely sampled relative to their chaining-probability to ensure a uniform selection within the respective intervals. Then, the probability  $\Pr\{t - j \leq \Delta_j^\phi \mid t_j \in M\}$  represents the proportion of the interval  $\mathbb{I}(j, i)$  "to the left" of  $j \leq \Delta_j^\phi$ . If both  $t$  and  $t_j$  were randomly selected variables from a perfect uniform distribution, it would follow:

$$\text{Uniform} \implies \Pr\{t - j \leq \Delta_j^\phi \mid t_j \in M\} = \frac{|\mathbb{I}_{j|j}|}{|\mathbb{I}_{j|i}|} = \frac{\Delta_j^\phi}{\Delta_i^\phi} = \frac{\lceil \phi j \rceil}{\lceil \phi i \rceil} \quad (7.148)$$

Further, notice how we can *not* assume that both (overlapping) intervals have equally likely probability masses since we are sampling inversely to the chaining likelihood. Therefore, the total probability of insertion at  $t_j$  is:

$$\Pr\{t_j \in M\} \times \Pr\{t - j \leq \Delta_j^\phi \mid t_j \in M\} = \frac{\mathbb{E}|Q_\phi^{\{j\}}|}{\Delta_j^\phi} \frac{\Delta_j^\phi}{\Delta_i^\phi} = \frac{\mathbb{E}|Q_\phi^{\{j\}}|}{\lceil \phi j \rceil} \frac{\lceil \phi j \rceil}{\lceil \phi i \rceil} = \frac{\mathbb{E}|Q_\phi^{\{j\}}|}{\lceil \phi i \rceil} \quad (7.149)$$

Hence, notice how the probability of being re-chained (i.e., *not* being replaced) is:

$$\Pr\{t - j > \Delta_j^\phi \mid t_j \in M\} = 1 - \frac{\Delta_j^\phi}{\Delta_i^\phi} = \frac{\Delta_i^\phi - \Delta_j^\phi}{\Delta_i^\phi} = \frac{\lceil \phi i \rceil - \lceil \phi j \rceil}{\lceil \phi i \rceil} \quad (7.150)$$

and because  $\lceil \phi i \rceil \geq \lceil \phi j \rceil$  due to  $j \leq i$ , the probability is zero if and only if  $\lceil \phi i \rceil = \lceil \phi j \rceil$ ; i.e. is removed *automatically* once it is out of the quantile range (which is the case when the next window size is equal the previous one).

Using the law of total probability, we verify that indeed the probabilities are consistent and correct:

$$\begin{aligned}
1 &= \Pr\{t_j \notin M\} + \Pr\{t_j \in M\} \\
&\iff \Pr\{t_j \notin M\} = 1 - \Pr\{t_j \in M\} \\
&= 1 - \left[ \Pr\{t_j \in M\} \times \Pr\{t - j \leq \Delta_j^\phi \mid t_j \in M\} + \Pr\{t_j \in M\} \times \Pr\{t - j > \Delta_j^\phi \mid t_j \in M\} \right] \\
&= 1 - \left[ \frac{^{\mathbb{U}}|Q_\phi^{\{j\}}|}{\lceil \phi i \rceil} + \frac{^{\mathbb{U}}|Q_\phi^{\{j\}}|}{\lceil \phi j \rceil} \frac{\lceil \phi i \rceil - \lceil \phi j \rceil}{\lceil \phi i \rceil} \right] \\
&= 1 - \left[ \frac{^{\mathbb{U}}|Q_\phi^{\{j\}}|}{\lceil \phi i \rceil} + \frac{^{\mathbb{U}}|Q_\phi^{\{j\}}|}{\lceil \phi j \rceil} \frac{\lceil \phi i \rceil - \lceil \phi j \rceil}{\lceil \phi i \rceil} \right] \\
&= 1 - \left[ \frac{^{\mathbb{U}}|Q_\phi^{\{j\}}| \lceil \phi j \rceil}{\lceil \phi i \rceil \lceil \phi j \rceil} + \frac{^{\mathbb{U}}|Q_\phi^{\{j\}}| \lceil \phi i \rceil - ^{\mathbb{U}}|Q_\phi^{\{j\}}| \lceil \phi j \rceil}{\lceil \phi i \rceil \lceil \phi j \rceil} \right] \\
&= 1 - \frac{^{\mathbb{U}}|Q_\phi^{\{j\}}| \lceil \phi j \rceil + ^{\mathbb{U}}|Q_\phi^{\{j\}}| \lceil \phi i \rceil - ^{\mathbb{U}}|Q_\phi^{\{j\}}| \lceil \phi j \rceil}{\lceil \phi i \rceil \lceil \phi j \rceil} \\
&= 1 - \frac{^{\mathbb{U}}|Q_\phi^{\{j\}}| \lceil \phi i \rceil}{\lceil \phi i \rceil \lceil \phi j \rceil} = 1 - \frac{^{\mathbb{U}}|Q_\phi^{\{j\}}|}{\lceil \phi j \rceil} = \Pr\{t_j \notin M\}
\end{aligned} \tag{7.151}$$

Also, note that  $^{\mathbb{U}}|Q_\phi^{\{j\}}| \leq k$ , but almost always  $^{\mathbb{U}}|Q_\phi^{\{j\}}| = k$ , particularly when  $k$  is small or  $i$  is large. If an item  $e_j$  is present in the reservoir at time-step  $j$ , it will remain there until it is replaced in its next chained step  $t_j$ . We will represent this as  $\downarrow_j^{i=t_j}$ , indicating that at time-step  $j$ , item  $e_i$  is chosen as the replacement for item  $e_j$ :

$$\Pr\{t_j \in M \wedge \downarrow_j^{i=t_j}\} = \Pr\{t_j \in M\} \times \Pr\{\downarrow_j^{i=t_j} \mid t_j \in M\} = \frac{^{\mathbb{U}}|Q_\phi^{\{i\}}|}{\Delta_j^\phi} \frac{1}{^{\mathbb{U}}|Q_\phi^{\{i\}}|} = \frac{1}{\Delta_j^\phi} \tag{7.152}$$

The survival probability that an item  $e_j$  which has been inserted but not (yet) removed at a current time-step  $i > j$  will now also *not* be replaced in the next time-step, and thus re-chained to a new index  $t$ , is:

$$\begin{aligned}
\Pr\{e_j \in Q_\phi^{\{i\}} \mid e_j \in Q_\phi^{\{i-1\}}\} &= 1 - \left( \Pr\{i \in M \wedge \downarrow_j^{i=t_j}\} \times \Pr\{t - j \leq \Delta_j^\phi\} \right) \\
&= 1 - \left( \frac{1}{\Delta_j^\phi} \times \frac{\Delta_j^\phi}{\Delta_i^\phi} \right) = 1 - \frac{1}{\Delta_i^\phi} = 1 - \frac{1}{\lceil \phi i \rceil}
\end{aligned} \tag{7.153}$$

However, we perform re-chaining repeatedly until substitution, so the above formula is not entirely corresponding. Each incoming chain index either replaces the previous one or is re-chained to a future one. To ensure continuous interval overlap without reusing "used" probability mass, the index of the re-chained index is always used as the reference. Let  $\tau_j$  denote the current (non-fixed, last) chain index for an index  $j$ . Initially,  $\tau_j \leftarrow j$ . Therefore, the probability that a given chain index is not replaced, but re-chained, given the last chain index, is:

$$\Pr\{t - j > \Delta_{\tau_{|j}}^\phi\} = 1 - \frac{\Delta_{\tau_{|j}}^\phi}{\Delta_i^\phi} = \frac{\Delta_i^\phi - \Delta_{\tau_{|j}}^\phi}{\Delta_i^\phi} \quad (7.154)$$

Again, notice how the probability is zero if and only if  $\Delta_i^\phi = \Delta_{\tau_{|j}}^\phi$ . Hence, the probability  $i = t_{\{\tau_{|j}\}}$ , that an index is selected as the subsequent chain index for a specific  $j$ , is:

$$\Pr\{i = t_{\{\tau_{|j}\}}\} \equiv \Pr\{i - j > \Delta_{\tau_{|j}}^\phi\} \times \frac{1}{\Delta_i^\phi - \Delta_{\tau_{|j}}^\phi} = \begin{cases} 0 & \text{if } \Delta_i^\phi = \Delta_{\tau_{|j}}^\phi \\ \frac{1}{\Delta_i^\phi} & \text{otherwise} \end{cases} \quad (7.155)$$

Thus the total survival probability  $\Psi_{|i}(e_j) := \Pr\{e_j \in Q_\phi^{\{[j+1,i]\}}\}$  of remaining in the reservoir from time-step  $j+1$  up to  $i > j$  is:

$$\begin{aligned} \Psi_{|i}(e_j) &= \prod_{i \geq j' > j} \left( 1 - (\Pr\{j' = t_{\{\tau_{|j}\}}\} \times \Pr\{j' - j \leq \Delta_{\tau_{|j}}^\phi\}) \right) \\ &\equiv \prod_{i \geq j' > j} \left( 1 - \begin{cases} 0 & \text{if } \Delta_i^\phi = \Delta_{\tau_{|j}}^\phi \\ \frac{1}{\Delta_{j'}^\phi} \times \frac{\Delta_{\tau_{|j}}^\phi}{\Delta_{j'}^\phi} & \text{otherwise} \end{cases} \right) \\ &= \begin{cases} 0 & \text{if } \Delta_i^\phi = \Delta_{\tau_{|j}}^\phi \\ \prod_{i \geq j' > j} \frac{(\Delta_{j'}^\phi)^2 - \Delta_{\tau_{|j}}^\phi}{(\Delta_{j'}^\phi)^2} & \text{otherwise} \end{cases} \end{aligned} \quad (7.156)$$

And so, given current index  $i$ , the maximum life-span interval for a selected index  $j$  is the interval  $[j+1, j + \Delta_i^\phi]$ , since it always holds  $\Delta_i^\phi \neq \Delta_{\tau_{|j}}^\phi$ , and therefore:

$$\Psi_{|j+\Delta_i^\phi}(e_j) = \prod_{j+\Delta_i^\phi \geq j' > j} \frac{(\Delta_{j'}^\phi)^2 - \Delta_{\tau_{|j}}^\phi}{(\Delta_{j'}^\phi)^2} \quad (7.157)$$

This formula tells us that, if  $i$  is small, the expected life-span of an index  $j$  (i.e. always being re-chained), is much smaller than when  $i$  is large, since  $\lim_{j' \rightarrow \infty} \frac{(\Delta_{j'}^\phi)^2 - \Delta_{\tau_{|j}}^\phi}{(\Delta_{j'}^\phi)^2} \approx 1$ ; which means that for large  $i$ , the life-span is about  $\Delta_i^\phi$  (linearly to the quantile window); but initially, it is not.

Overall, as a general formula, the total probability  $p_{[j,i]}$  that an item  $e_j$  is in the reservoir at a time-step  $i \geq j$  is the product of the probability of it being included in the reservoir initially and the probability of it not being replaced until time-step  $i$ . The removal of expired items is guaranteed because the product becomes zero if a term is zero, which occurs when it lies outside the quantile window:

$$p_{[j,i]} \triangleq \frac{\Psi_{|i}(e_j)}{\Delta_j^\phi} \times \begin{cases} 1 & \text{if } j = i \\ \Psi_{|i}(e_j) & \text{otherwise} \end{cases} \quad (7.158)$$

### 7.16.2 Full Quantile-Sampling

The full quantile-sampling algorithm for a trajectory is then the following:

**Algorithm 3:** Representative Trajectory Sampling (RT-Sampling)

---

**Input** :  $T$  (trajectory stream)  
**Require**:  $0 < \phi \leq 1$  (quantile),  $k$  (max. size per reservoir)  
**Output** : Trajectory samples (ordered by time-step)

**Function** Balance( $\Xi, Q$ ):  
 Check if any element was *added* or *deleted* in the last update of  $Q$ ;  
**if** *deleted*  $(\vec{z}_j, t_j)$  **then**  
   Delete storage reference  $(Q, j) \mapsto (\vec{z}_j, t_j)$  from  $\Xi$ ;  
**if** *added*  $(\vec{z}_i, t_i)$  **then**  
   Insert storage reference  $(Q, i) \mapsto (\vec{z}_i, t_i)$  into  $\Xi$ ;

Initialize FIFO-queue  $Q_{\text{top}}$  (max. size  $k$ );  
 $Q_\phi \leftarrow$  New empty quantile-reservoir (max. size  $k$ );  
Initialize self-balancing data structure  $\Xi$  of storage references (balanced on time-steps);  
**while**  $\text{new } (z_i, t_i) \in T; i \in \mathbb{N}^+$  **do**  
 $e_i \leftarrow (z_i, t_i)$ ;  
 FIFO-Update  $(e_i, Q_{\text{top}})$  & Balance( $\Xi, Q_{\text{top}}$ );  
 DQCS-Update  $(e_i, \phi, k, Q_\phi)$  & Balance( $\Xi, Q_\phi$ );  
**yield**  $\Xi.\text{DATA}()$ ;

---

The RT-Sampling algorithm is designed to handle a continuous trajectory stream by ensuring that representative samples are maintained at any time (i.e. time-step). It accomplishes this by utilizing a FIFO queue to track recent entries, effectively creating a sliding window, and employing a quantile array to keep representative samples within a specified quantile range (over the complete data stream history so far; without actually storing the entire trajectory).

To efficiently handle the random time-step order across reservoirs, RT-Sampling uses a self-balancing data structure  $\Xi$ . This structure maintains ordered reference pointers for stored elements, facilitating real-time additions and deletions as new data arrives and older data expires. Storing references to item locations in the reservoir as pointers ensures efficient retrieval at any given time. Notice here how, ultimately, the goal is to also maintain a *chronological* sampled trajectory "efficiently". Naively sorting the entire reservoir each time anew is *not* efficient.

**Take-Away**

A decaying quantile chain-sampling algorithm can be designed for sampling from continuous (infinite) data streams, such as spatial-temporal trajectories, where the goal is to maintain a real-time sample within a quantile range of a size based on the stream's current length. Keeping the quantile sample within memory constraints, representative, and including only valid, non-expired items, is challenging. The term "representative" in this context can be mathematically defined, e.g. by examining, establishing, and ensuring (specific) mathematical properties.

**Keywords**

Dynamic Quantile Window • Chain-Sampling • Reachability Likelihood



## Chapter 8

# Experimental Methods

### 8.1 Hardware

Training DNNs typically requires significant computational resources, relying on GPUs, TPUs, or specialized AI hardware accelerators instead of CPUs. This extends beyond just training and applies to evaluation and inference stages as well, although, for inference, more specific hardware solutions can be utilized due to the absence of back-propagation calculations. DNN training, thus, involves intensive iterative calculations, including large matrix multiplications, back-propagation operations, and other optimization algorithms. This computational complexity is, in particular, effectively managed through parallel processing.

Choosing the right DL hardware depends on several factors such as model and dataset size, desired training duration, cost considerations associated with (parallel) training, etc. Hence, we utilized a diverse array of different GPU hardware (types and quantities), matched to the specific requirements of each model, training objectives, and use cases.

#### Take-Away

Training neural networks usually demands substantial computational power, requiring the use of GPUs, TPUs, or dedicated AI hardware accelerators instead of CPUs.

#### Keywords

Hardware • Deep Neural Networks • GPUs • TPUs • Computational Power

### 8.2 High Dimensional Data Visualization

Visualization is often used to cross-check results, gain a better intuition and understanding of the results, and compare different outcomes quickly and easily. However, visualizations are *no* proofs, nor do they validate any empirical findings; rather, they aid in "supporting" theories. In the context of ML, it is often advantageous to visualize complex high-dimensional data. However, because visualization may only be done in two or (at most) three dimensions, high-dimensional data must here be compressed (e.g. projected). Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) (McInnes, Healy, and Melville, 2018) appeared as a practical and scalable and new manifold learning methodology for dimensionality reduction. Currently, UMAP is frequently employed as the primary option for data and embedding visualization.

UMAP received a lot of attention, and it was even noted and described by Google researchers, who created a very famous interactive blog post called Andy Coenen, 2019, in which they, among other things, underlined the following key considerations when working with UMAP visualizations:

- Hyperparameters matter.
- Cluster sizes in the UMAP plot mean nothing.
- Distances between clusters might not mean anything.
- Random noise does not always look random.
- UMAP algorithm is stochastic.

Figure 8.1 gives an example of the stochastic property of UMAP where visualizations on the same data yield different results on each run.

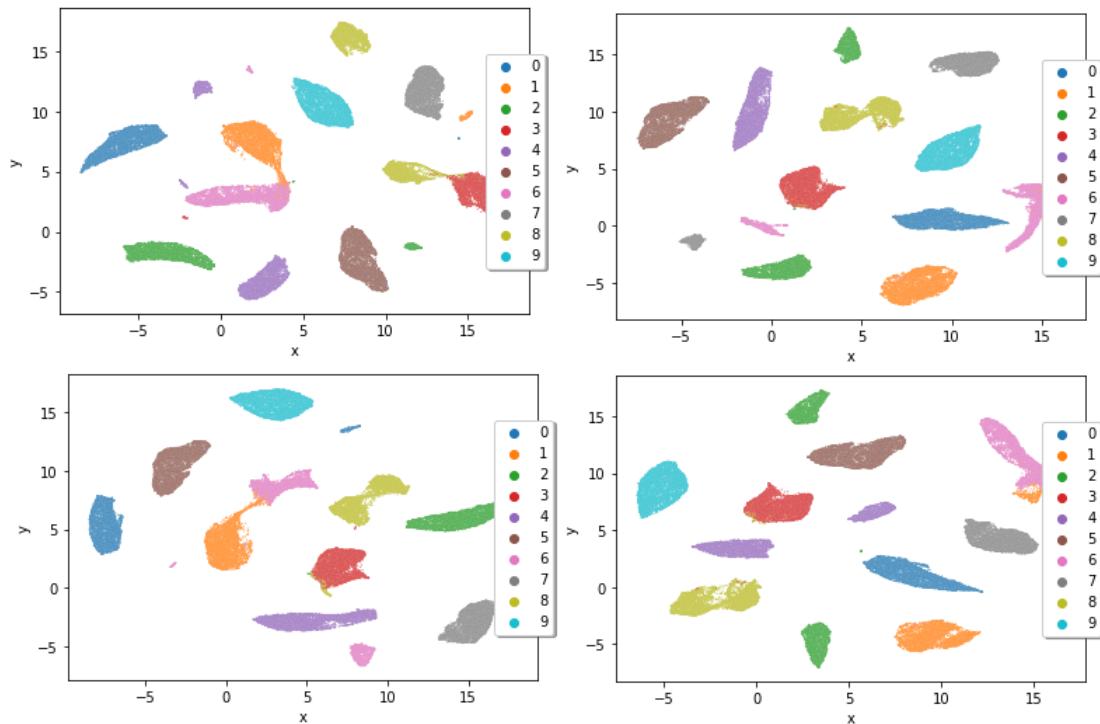


FIGURE 8.1: An example of how multiple visualizations using UMAP on the same data may provide different outcomes.

### Take-Away

Visualizations are a powerful tool, but should always be taken with a grain of salt, keeping inherent limitations and drawbacks in mind.

### Keywords

Visualization • High Dimensional Data • Dimensionality Reduction • UMAP

### 8.3 Adaptive Complex-Valued Bi-Nonlinear Neural Networks

**Notice:** Our work and findings on complex-valued NNs (presented in this thesis) have been published in Guimerà Cuevas, Phan, and Schmid, 2023. It was inspired and builds upon our previous research (Guimerà Cuevas and Phan, 2021), but differs significantly as it has been substantially improved and modified (both the complex architecture and its mathematical framework); thus, constitutes novel contributions. In particular, this novel contribution has been accepted for publication, after peer review. The Version of Record is available online at: [https://link.springer.com/chapter/10.1007/978-3-031-33374-3\\_28](https://link.springer.com/chapter/10.1007/978-3-031-33374-3_28). Use of this Version is subject to the publisher's Manuscript terms of use. Therefore, please refer to Guimerà Cuevas, Phan, and Schmid, 2023 for specific details.

### 8.4 Adaptive Tanh-Normalization

**Please note:** This contribution has been accepted for publication, after peer review in Guimerà Cuevas and Schmid, 2024b. The Version of Record is available online.

We used several common statistical distributions for analysis, including an Alpha-Gamma distribution with parameters  $\alpha = \gamma = 4.0$ , a generalized Pareto distribution with parameter  $\xi = 1$ , a symmetric bi-modal distribution with parameters location=-3, scale=1, size=10, etc. We further created an artificial synthetic classification set of size 10,000 with ten classes. Each class in the synthetic set consisted of two Gaussian distributed clusters spanning along the vertices of a hyper-cube. To introduce covariance, features were independently drawn and then randomly (linearly) combined within each cluster. Outliers are restricted to mixtures of Gaussian distributions, which may be a limitation here. Additionally, 5% of the labels were randomly assigned, introducing multivariate outliers. A small feed-forward neural network with four hidden layers, each containing 64 neurons, was used to classify the synthetic data. The Gaussian Error Linear Unit was chosen as the activation function. Synthetic data sets are useful for benchmarking because they offer control of data properties and allow quick and deliberate performance testing. WTE was also evaluated on four commonly used toy computer vision datasets (Fashion-MNIST, EMNIST, CIFAR10, and CIFAR100) using ResNet9. Pixel values were normalized after applying a gray-scale conversion; images were re-scaled to a size of 32x32. We used classification performance over time as the evaluation metric. As a real-world dataset, CDC Diabetes Health Indicators (Teboul, 2023) was used. For a fair and accurate evaluation, each training iteration involved initializing the model weights identically for all different FS methods. That is, model weight initialization was identical for all methods within a training run but different between different runs. The results were averaged across five runs. Cosine annealing was used as the learning rate scheduler, starting at 0.001.

### 8.5 Determining Representative Textual Labels for Clustering Accurate Sensor Data with Inexact Annotations

Experiments are evaluated on the COCO-2017 (Lin et al., 2014) and TextCaps (Krause et al., 2017) annotated image datasets. Both are large benchmarks with diverse images and multiple descriptions per image. We experimented with raw unprocessed captions for COCO-2017 and paraphrased preprocessed captions on TextCaps. During training, the target caption was randomly selected each time. The ViED model

was initialized with a pre-trained SWINV2 (Liu et al., 2022) as the encoder and GPT2 (Radford et al., 2019) as the decoder, connected through cross-attention. The ViED model was fine-tuned on each dataset, respectively. Clusters were obtained via HDBSCAN (Campello, Moulavi, and Sander, 2013), a density-based clustering algorithm, with a minimum cluster size of 15 to avoid small clusters. Our approach is versatile, as it is not tied to specific clustering algorithms, encoders, or decoders, and it is modality-independent. It can be applied to any-to-text modality, not strictly or necessarily limited to images.

As text similarity metrics, we used BERT-Score with "distilbert-base-uncased" (Sanh et al., 2019; Zhang\* et al., 2020) as the base model, BART-Score with "distilbart-cnn-12-6" (Yuan, Neubig, and Liu, 2021), as well as BLEURT with "BLEURT-20-D6" (Sellam, Das, and Parikh, 2020). Similarity between CCs and reference texts was determined by the average score. Distilled models were used to speed up the evaluation procedure. We chose BERT, BART, and BLEURT since they are famous text similarity metrics that use transformer models to capture nuanced semantics and context and are effective for evaluating generated texts.

We compared various basic pooling techniques, including the mean, median, geometric median (GM), medoid, maximum, and random pooling (vector sampled from a normal distribution). We explored various text generation strategies, including greedy-search,  $k$ -sequence sampling, multinomial sampling, beam-search, beam-search multinomial sampling, and diverse beam-search (Wolf et al., 2020b). These strategies yield varying degrees of diversity and coherence in the generated text. Additionally, as a reference, the maximum and average intra-cluster similarity of all original labels was determined (brute-force). For performance reasons, we here randomly selected one caption from each instance as a candidate but still evaluated against all the (multiple) target captions of all cluster instances.

### 8.5.1 Quality Assessment of Textual Generations

A modification to BERT-score is the *re-scaled* BERT-Score (Zhang\* et al., 2020). It has been widely accepted in the natural language processing community as a reliable and robust measure for evaluating text generation models as a metric of quality assessment. BERT-score itself uses Roberta-Large as the base model. It is known to provide strong assertive performance on a wide range of natural language understanding tasks, and its ability to provide a single, scalar score for each generated text as a measure of text similarity. Additionally, BERT-Score has been found to be computationally efficient and able to handle variations in text length.

As stated by the authors, the re-scaled BERT-Score is a modification of the original BERT-Score that aims to improve its interpretability by introducing a re-scaling factor. The original BERT-Score calculates a sentence-level similarity score by using cosine similarities between the contextual embeddings of the tokens in the reference text and the generated text. The numerical range of the original BERT-Score is between -1 and 1, in accordance with the underlying cosine similarity. To make the BERT-Score more interpretable, a baseline value is subtracted from the original BERT-Score, resulting in a more intuitive numerical range. This baseline value is obtained by averaging the BERT-Scores of a large number of random sentence pairs from a specific corpus. The re-scaled BERT-Score preserves the correlation with human judgments as measured by Pearson's and Kendall's coefficients. However, it is important to note that the re-scaling process is not guaranteed to have a range between 0 and 1; but is typically the case. Re-scaling is done using the following simple linear adjustment on a baseline  $\epsilon$ :

$$\hat{x} = \frac{x - \epsilon}{1 - \epsilon} \quad (8.1)$$

The re-scaled BERT-score's default baseline  $\epsilon$  is empirically determined based on the Common Crawl dataset (Wenzek et al., 2019).

## 8.6 Model Output Calibration & Novelty Detection

**Please note:** This contribution has been accepted for publication after peer review in Guimerà Cuevas and Schmid, 2024a and has been presented in the conference proceedings.

For evaluation, we focused on DL scenarios that produce both: embedding representations and classification probabilities. We took three approaches: (1) We fine-tuned a pre-trained ResNet18 model (He et al., 2016) on a specific dataset (inliers) and forwarded a different OOD dataset (outliers) through the model. The goal was to identify elements as either an inlier or an outlier, respectively. The penultimate layer vector representation was used as the feature representation for the baseline algorithms. (2) Given one dataset, we trained only half of the classes and regarded the other half as outliers. (3) We conducted experiments against synthetically generated data in form of cluster groups (inliers) and uniformly sampled points outside the clusters (outliers). Here the raw features (instead of deep embeddings) were used by the baseline outlier detection algorithms.

All results represent the average of 15 independent runs. We used the score adjusted formula 7.130 (with  $\lambda_1 = 2, \lambda_2 = k - 2$ ) and compared it against multiple different common outlier detection algorithms (Zhao, Nasrullah, and Li, 2019; Han et al., 2022): ECOD, SAMPLING, QMCD, ABOD, OCSVM, MCD, COF, LOF, HBOS, LODA, IFOREST, INNE, SUOD, HDBSCAN; as well as against an ensemble method which was the maximum unified score (Kriegel et al., 2011) from these models. Additionally, we considered hybrid models, which represented the highest combined score achieved by pairing a particular baseline algorithm with our proposed OOD approach (for all baselines respectively). Evaluation was done using the "Area Under the Curve" (AUC) metric since it works well for binary classification problems (outlier or not); measuring the entire area underneath the ROC curve from (0,0) to (1,1). AUC provides an aggregated measure of performance across all possible classification thresholds. We chose BetaCalibration over Isotonic regression to prevent plateaus; and performed percentile-wise calibration using  $\Xi_{\hat{y}}^{\Phi}(p_i^{[j]})$ .



## Chapter 9

# Analysis & Results

## 9.1 Adaptive Complex-Valued Bi-Nonlinear Neural Networks

**Notice:** Our work and findings on complex-valued NNs (presented in this thesis) have been published in Guimerà Cuevas, Phan, and Schmid, 2023. It was inspired and builds upon our previous research (Guimerà Cuevas and Phan, 2021), but differs significantly as it has been substantially improved and modified (both the complex architecture and its mathematical framework); thus, constitutes novel contributions. In particular, this novel contribution has been accepted for publication, after peer review. The Version of Record is available online at: [https://link.springer.com/chapter/10.1007/978-3-031-33374-3\\_28](https://link.springer.com/chapter/10.1007/978-3-031-33374-3_28). Use of this Version is subject to the publisher's Manuscript terms of use. Therefore, please refer to Guimerà Cuevas, Phan, and Schmid, 2023 for specific details on the following:

### 9.1.1 XOR-Problem

### 9.1.2 Minimal Networks & Expressive Power

### 9.1.3 Classification

#### Take-Away

An adaptive bi-nonlinear layer design can effectively mitigate extreme neural weights by maintaining weights within a unit circle, thus preventing excessive weight dominance or suppression of other neural inputs. The complex-valued layers exhibit effectiveness, particularly at high learning rates, and offer improved problem-solving neural expressiveness; they can solve the XOR problem. However, the forward pass of the model requires dual matrix multiplications due to the real and imaginary components.

#### Keywords

Bi-Nonlinear Layers • Complex-Numbers • Unit Circle • Learning Rates • Expressiveness & Effectiveness

## 9.2 Loss-Weighting

To account for a more robust, yet dynamic loss weighting, we mentioned that one approach is to calculate the dynamic weight values for the loss functions by centering the loss values around one (Equation 7.38). If  $|L|$  denotes the number of losses, and  $\mathbb{E}$  the mean average, then we know that:

$$\begin{aligned}
\mathbb{E}[f(x)] &:= \mathbb{E}\left[1 + \frac{x}{\sum_{0 \leq i < |\mathbf{L}|} x_i} - \frac{1}{|\mathbf{L}|}\right] \\
&= 1 + \mathbb{E}\left[\frac{x}{\sum_{0 \leq i < |\mathbf{L}|} x_i}\right] - \frac{1}{|\mathbf{L}|} \\
&= 1 + \frac{1}{|\mathbf{L}|} \left[ \sum_{0 \leq i < |\mathbf{L}|} \frac{x_i}{\sum_{0 \leq j < |\mathbf{L}|} x_j} \right] - \frac{1}{|\mathbf{L}|} \\
&= 1 + \frac{1}{|\mathbf{L}|} \left[ \frac{\sum_{0 \leq i < |\mathbf{L}|} x_i}{\sum_{0 \leq j < |\mathbf{L}|} x_j} \right] - \frac{1}{|\mathbf{L}|} \\
&= 1 + \frac{1}{|\mathbf{L}|} - \frac{1}{|\mathbf{L}|} \\
&= 1
\end{aligned} \tag{9.1}$$

i.e. the expected mean loss weight is one. As mentioned earlier, there are two approaches here:

1. **Adjusted weights:** Calculating the loss weights by modifying the Softmax of the losses. It adds one to the softmax output and subtracts the reciprocal of the total number of losses:

$$weights_{adj} = 1 + softmax(x) - \frac{1}{|\mathbf{L}|}$$

2. **Multiplied weights:** The standard method where the weights are calculated by multiplying the Softmax output by the total number of losses:

$$weights_{mul} = |\mathbf{L}| \times softmax(x)$$

The **Adjusted method** has the advantage of being less sensitive to outliers, but the disadvantage is that it might underweight significant losses. On the other hand, the **Multiplicative method** maintains the original proportion of the softmax output and is so more sensitive to the nuances of the data. However, this becomes a drawback in the presence of outliers, as it is more sensitive to them. This can be seen by visualizing the histogram distribution and their standard deviation with increasing  $|\mathbf{L}|$ . Overall, choosing between both methods depends on the data and the use case.

Figure 9.1 illustrates the histogram distributions of individual loss weights of 10.000 vectors for different numbers of losses ( $|\mathbf{L}|$  be here referred to as the "loss dimensionality"). The std. graph for increasing  $|\mathbf{L}|$  is shown in Figure 9.2.

### Take-Away

Traditional loss weighting does not produce an expected mean loss weight of one, but rather  $1/|\mathbf{L}|$ . However, we can enforce a mean of one by rewriting the formula over a simple linear transformation.

### Keywords

Loss Weighting • Linear Transformation • Expected Mean • Variance

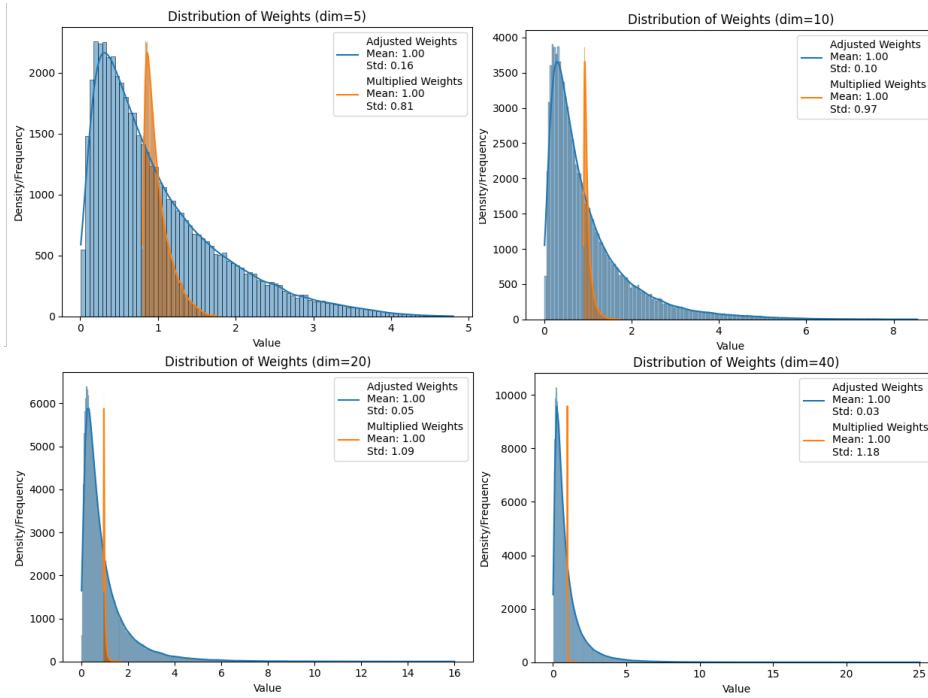


FIGURE 9.1: Comparing the distribution of adjusted and multiplied weights derived from random losses.

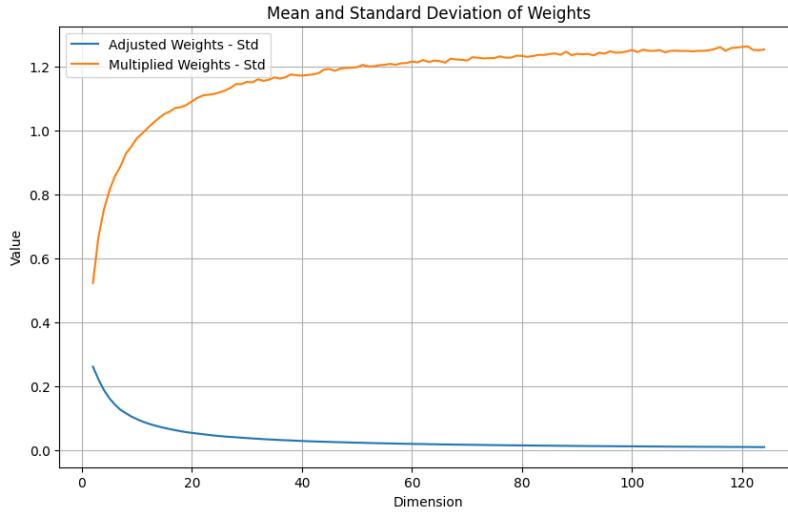


FIGURE 9.2: Comparing the distribution of adjusted and multiplied weights derived from random losses.

## 9.3 Embeddings

### 9.3.1 Clustering Representations

To gain initial insights into the potential clustering quality, we can contrast our clustered representation with that of randomly clustered points and observe key differences. Overall, it is well-known that clustering becomes progressively more difficult in higher dimensions.

Let  $M \in \mathbb{R}^{m \times n}$  be a matrix that contains  $m$  random vectors of dimension  $n$ . We can then apply a clustering algorithm (e.g. K-Means) to the rows of  $M$  and visualize

the clustering results by projecting the vectors and their clusters (i.e. labels) into a two-dimensional plane using principal component analysis (PCA). If we analyze the spatial distribution of all points and cluster centroids, we see that the centroids tend to condense towards the center with increasing dimensionality  $n$ , and thus the cluster groups gradually become more mixed and disordered, and eventually visually hard to separate from one another (Figure 9.3). Recall that in high-dimensional spaces, randomly drawn vectors tend to be increasingly orthogonal to one another as the dimensionality increases.

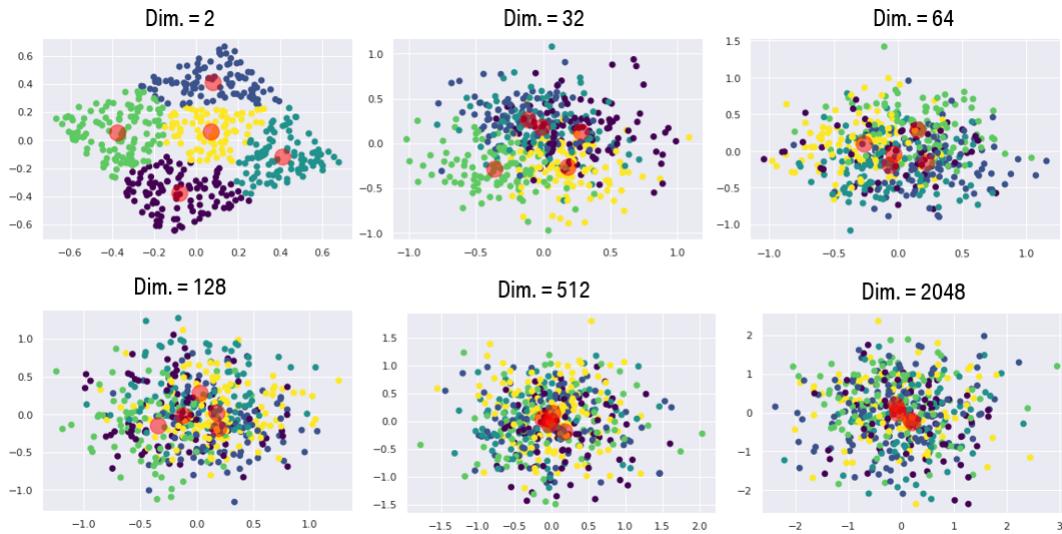


FIGURE 9.3: Projecting the clustering results of random vectors using PCA onto a two-dimensional plane. The colors of the points indicate the labels, whereas the cluster centroids are depicted as big red circles.

### Take-Away

Clustering becomes increasingly challenging as dimensions increase.

### Keywords

- Clustering • Random Points • Matrix Representation • Dimensionality

### 9.3.2 Hierarchical Dependent Embedding Representations

In a hierarchical tree structure where nodes contain feature representations and features of their subtrees, it is already intuitive that mistakes occurring higher in the structure carry greater impact than those lower down. For instance, an error in the feature representation at the topmost (root) node holds more weight and consequence than an error found at a lower-level node, such as a leaf node. That is, the impact of errors or inaccuracies within a hierarchical relational structure varies depending on their specific location within the hierarchy. And so, since the top root node embedding summarizes (i.e. represents) the entire structure, any mistake here severely misrepresents and considerably influences the overall understanding (i.e. encoding) of the tree. Errors in lower-level nodes, however, affect smaller portions and have less impact on the overall structure.

To mathematically quantify this error, let  $\epsilon$  represent the percentage of (expected) information error in embedding a node's subtree. For instance,  $\epsilon = 5\%$  indicates 5% information error during the encoding process into a single vector representation. Let  $\epsilon_\theta$  be the error at node  $\theta$ ;  $\omega \prec \theta$  denotes that  $\omega$  is a **direct** child-node of  $\theta$ , and  $\perp$  indicates a leaf node.

One approach to quantifying the severity of the information error involves considering the cumulative impact of errors at different hierarchical levels by focusing on the integrity of the information throughput  $\eta := 1 - \epsilon$ :

$$\mathcal{I}_\perp := \eta_\perp \quad \text{and} \quad \mathcal{I}_\theta := \eta_\theta \cdot \mathbb{E}[\mathcal{I}_{\omega \prec \theta}] \quad (9.2)$$

If we assume a constant error  $\epsilon$ , then  $\eta$  is also constant, and so:

$$\mathbb{E}[\mathcal{I}_{\omega \prec \theta}] = \eta^{\#level-1} \implies \mathcal{I}_\theta = \eta^{\#level} \quad (9.3)$$

The deviation from unity then represents the expected information error:

$$\mathcal{L}_\theta = 1 - \mathcal{I}_\theta \quad (9.4)$$

and so, the total error is given by  $\mathcal{L}_{root}$ . Overall, this allows us to quantify the error (and the integrity) numerically and compare it to different model configurations. In practice, the error rate may not be constant (i.e. equal) for all levels. If we have limitations on computational power, this helps us optimize the distribution of computational resources to achieve the highest embedding quality  $\mathcal{I}_{root}$ . Determining the corresponding  $\epsilon$  values can be tricky, yet employing different metrics or heuristics such as the level-wise AE reconstruction- or classification losses can help.

Notice that the primary goal or intention may not necessarily be to preserve *all* input information. Certain details may not be relevant or required for achieving specific (training) objectives, and hence may not contribute to (or be reflected in) the semantic manifold. In such cases, the concept of information "error" (which could be misunderstood as information "loss") can be better reframed as information "distortion", which may convey the essence of the situation more accurately. The formula here, however, remains unchanged. The fundamental key conceptual difference lies in that we are not trying to preserve all input data but preserve all relevant "characteristic" information. So in other words, distortion or error refers to inadvertently modifying the optimal embedding representation intended for a particular objective.

### Large Errors at Root-Nodes (Upper Levels)

Errors occurring at a root node level have a severe and disproportionate impact on the entire respective hierarchical structure; since they significantly distort the representation of the whole tree structure; no matter how "good" the representation quality has been so far. In fact, let us assume a perfect encoding without any information error for all non-root nodes, and so  $\epsilon_{\neg root} := 0$ . Yet, for the root node, the error be  $\epsilon_{root}$ . Obviously, the final encoding error for the entire tree structure then equals  $\epsilon_{root}$ . This can be very problematic, even unacceptable, if  $\epsilon_{root}$  is large.

To illustrate this better, consider an extreme scenario where  $\epsilon_{root} = 100\%$ . In this case, despite achieving flawless encoding for all other subtrees, nodes, and features with perfect representations, the resulting final embedding of the root node, and thus the entire tree, would be entirely erroneous, inaccurate, and "random", because the extreme error at the root node causes a complete misrepresentation; rendering so the overall tree encoding essentially meaningless and, therefore, useless.

### Large Errors at Leaf-Nodes (Lower Levels)

Similarly, let  $\epsilon_{\perp}$  denote an error specifically at a leaf node, while  $\epsilon_{\neg\perp} := 0$  represents error-free encoding for non-leaf nodes. The impact of an error  $\epsilon_{\perp}$  at this (lower) level remains constrained within the subtree originating from that specific leaf node. It does not compromise the quality of other subtrees or nodes within the hierarchy. The error is still propagated upwards, but its influence is minimal (due to the expectation operation  $\mathbb{E}[\cdot]$ ); having so only a limited effect on nodes beyond its immediate hierarchical level.

### Implications

Given a fixed computational resource budget for ensuring embedding quality, it is more effective to prioritize higher-level nodes over lower-level ones, as well as nodes with a large number of sub-nodes in their subtree. Overall, the goal here is to minimize  $\mathcal{L}_{root}$ . A simple practical approach is to distribute resources according to the number of sub-children and the hierarchical level (the root being the highest level). This is consistent with the argument that error implications become less severe further down the hierarchy than higher up, and errors in nodes with fewer sub-nodes are less severe than errors on a node representation encapsulating many subnodes (a bigger portion of the tree would be misrepresented/distorted). As a result, focusing resources on higher-level nodes and those that contain larger subgraphs with more nodes is logical.

#### Take-Away

Errors at the root node have a far-reaching global impact, distorting the entire hierarchical representation; the distortion propagates across the entire structure. In contrast, errors at leaf nodes have local effects, influencing only the subtree rooted at that particular leaf without significantly altering the broader tree's overall representation.

#### Keywords

Root Node Errors • Hierarchical Representation • Propagation of Errors

## 9.4 Deep Learning for Hierarchical Databases with Recursive One-to-Many Relations

### 9.4.1 Unsupervised Deep Clustering

Figure 9.4 displays the (unsupervised) deep representations of the projected embeddings, as well as instances that are near together in the embedding space.

An interesting observation is that these instances were *not* grouped or organized (i.e. embedded) based on visual similarities, but rather solely by their components. Furthermore, even though we did *not* have information about how the individual parts are assembled into whole objects, the model still managed to find interesting and good similarities between the objects (even visual ones); encoding similar objects closely together in the embedding space. Again, remarkably, this was achieved purely based on the compositional hierarchical characteristics of the objects.

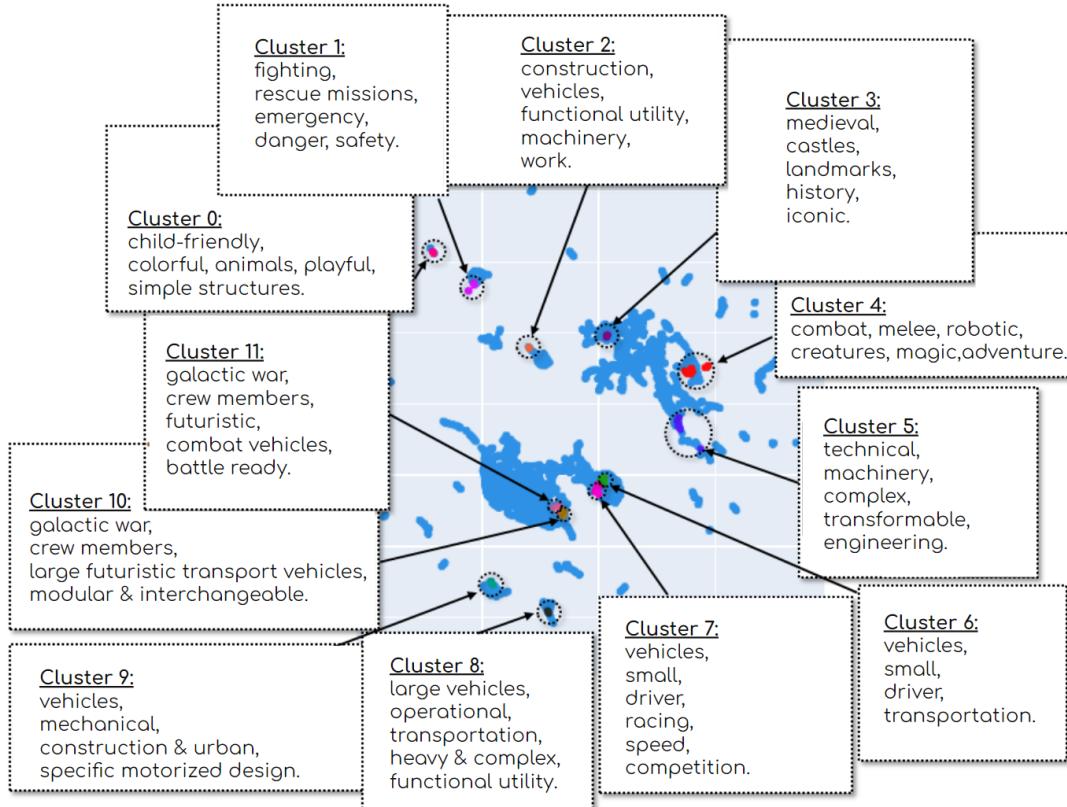


FIGURE 9.4: The embedding projection and the closest points of random instances within some clusters are visualized. The images of the instances are from the data set (Rebrickable<sup>®</sup>, 2022).

#### 9.4.2 Supervised Classification

A strictly supervised classification objective and a semi-supervised classification task with identity preservation (using AEs) were examined. Ablation tests were conducted on the real-world Rebrickable<sup>®</sup> set and on synthetic data, with kernel embeddings (KEs) and residual connections (RCs) turned on and off. Overall, using RC had the best supervised task performance. However, in the semi-supervised task with identity preservation, KE and RC demonstrated the best test set performance on the real-world data, and the best training convergence on synthetic data (Figure 9.5). Omitting both, RC and KE, resulted in the poorest test performance for the Rebrickable<sup>®</sup> set; but performance was good on the train set. KE + RC produced the best results on the test set but showed somewhat slower training convergence on the train set. The synthetic data set was similar: on the pure classification task, using RC achieved top performance; excluding KE and RC also on the train set, but produced the poorest results on the semi-supervised task, where the best training convergence was obtained by KE + RC (second-best convergence performance was using only RC). This suggests that RCs may be most effective for pure classification tasks, whereas using KEs + RCs may yield the best results for semi-supervised tasks requiring identity- or information preservation.

Moreover, we also evaluated performance by comparing it against a brute-force transformer-based approach directly applied to a textual representation of the data (sentences correspond to a forced textual encoding of individual root-to-leaf paths;

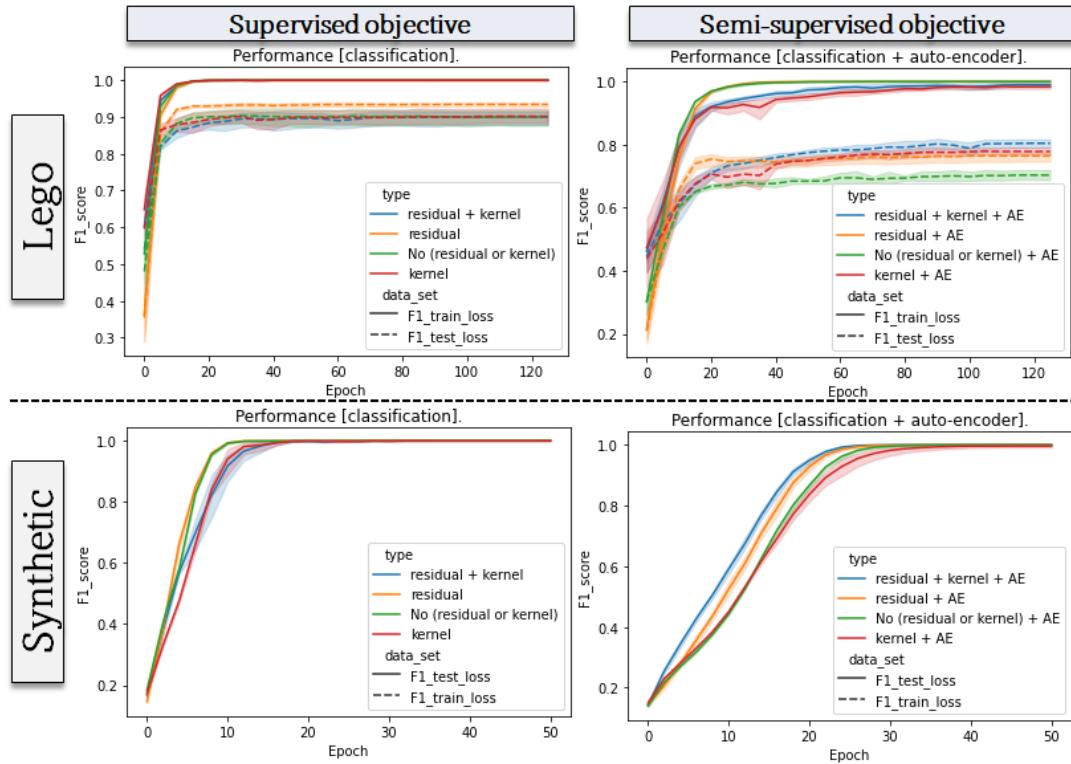


FIGURE 9.5: The results of a classification and semi-supervised task are displayed for the ten most frequent classes. The upper graphs relate to the Rebrickable® DB; the lower ones to the synthetic data.

entire objects represented as a collection of these paths/sentences), serving as a straightforward baseline. We evaluated the classification performance of two Bert models (mini, and small; Turc et al., 2019) and compared them with two versions of the HDB-model that, respectively, had a comparable number of trainable weights to Bert-mini and Bert-small. To evaluate their performance, we conducted experiments on a larger classification task consisting of 50 labels. We used the weight ratio of a model with respect to the baseline of Bert-mini ("model/BERT-mini") as  $\mu$  to denote the model size discrepancy. A value of  $\mu$  greater than one indicates that the corresponding model has more weights than Bert-mini; a value smaller than one that it has fewer weights. The weight ratios for the HDB-models were 1.01 and 3.64, while the ratios for the Bert models were 1.00 (mini) and 3.71 (small). The KE was excluded here as identity-preservation was not a relevant task (only classification); but residual connections were still used. The results are shown in Table 9.1 below. Because the models differed in architecture and size, we compare the best achieved average score for each model across multiple runs with different learning rates, trained for 25 epochs in total.

#### 9.4.3 Aggregation Function

Training occurs in batches consisting of instances represented as graphs. Due to variations in graph sizes, a simplistic approach involves padding to ensure uniform

TABLE 9.1: Class-weighted F1 classification scores of various models with different model sizes and learning rates.  $\mu$  represents the model size ratio to BERT-mini for a given model;  $\eta$  is the learning rate. Bert was used without positional encoding; token order was randomly sampled each time before truncation. The median sequence length was 234 (mean  $\sim$  367). Bert employed the default maximum sequence length of 512. However, to emulate a situation where the textual DB encoding exceeds the sequence length limit, a modified version with a limit of 128 (a fourth of the maximal sequence windows) was also evaluated; denoted by \*Bert.

model $[\mu]$	lr. $\eta$	Epoch											
		Train						Test					
		1	5	10	15	20	25	1	5	10	15	20	25
HDB-v1 $[\mu=1.01]$	1e-3	<b>0.71</b>	<b>0.91</b>	<b>0.97</b>	<b>0.99</b>	<b>1.0</b>	<b>1.0</b>	<b>0.62</b>	<b>0.7</b>	<b>0.72</b>	<b>0.74</b>	<b>0.75</b>	<b>0.75</b>
	1e-4	0.51	0.75	0.86	0.9	0.92	0.92	0.47	0.65	0.7	0.72	0.72	0.73
	2e-5	0.22	0.38	0.52	0.58	0.6	0.6	0.21	0.35	0.48	0.53	0.54	0.54
HDB-v2 $[\mu=3.64]$	1e-3	0.66	0.87	0.96	0.99	<b>1.0</b>	<b>1.0</b>	0.58	0.69	<b>0.73</b>	<b>0.75</b>	<b>0.75</b>	<b>0.76</b>
	1e-4	<b>0.74</b>	<b>0.96</b>	<b>0.99</b>	<b>1.0</b>	<b>1.0</b>	<b>1.0</b>	<b>0.63</b>	<b>0.72</b>	<b>0.73</b>	0.74	0.74	0.74
	2e-5	0.39	0.69	0.83	0.89	0.91	0.91	0.35	0.58	0.67	0.69	0.7	0.7
Bert-mini $[\mu=1.00]$	1e-3	0.02	0.03	0.02	0.03	0.02	0.02	0.02	0.03	0.02	0.03	0.02	0.02
	1e-4	<b>0.11</b>	<b>0.5</b>	<b>0.73</b>	<b>0.81</b>	<b>0.85</b>	<b>0.85</b>	<b>0.11</b>	<b>0.46</b>	<b>0.64</b>	<b>0.69</b>	<b>0.71</b>	<b>0.71</b>
	2e-5	0.02	0.15	0.27	0.35	0.38	0.38	0.02	0.15	0.26	0.33	0.36	0.36
Bert-small $[\mu=3.71]$	1e-3	0.01	0.02	0.02	0.02	0.02	0.01	0.01	0.02	0.02	0.02	0.02	0.01
	1e-4	<b>0.2</b>	<b>0.71</b>	<b>0.89</b>	<b>0.94</b>	<b>0.96</b>	<b>0.97</b>	<b>0.2</b>	<b>0.64</b>	<b>0.75</b>	<b>0.78</b>	<b>0.79</b>	<b>0.79</b>
	2e-5	0.02	0.38	0.59	0.68	0.72	0.72	0.02	0.37	0.55	0.63	0.66	0.66
*Bert-mini $[\mu=1.00]$	1e-3	0.01	0.02	0.02	0.02	0.02	0.02	0.01	0.02	0.02	0.02	0.02	0.02
	1e-4	<b>0.11</b>	<b>0.42</b>	<b>0.56</b>	<b>0.63</b>	<b>0.66</b>	<b>0.66</b>	<b>0.11</b>	<b>0.4</b>	<b>0.51</b>	<b>0.57</b>	<b>0.59</b>	<b>0.59</b>
	2e-5	0.02	0.13	0.25	0.31	0.34	0.34	0.02	0.13	0.24	0.3	0.32	0.32
*Bert-small $[\mu=3.71]$	1e-3	0.01	0.02	0.02	0.02	0.02	0.02	0.01	0.02	0.02	0.02	0.02	0.02
	1e-4	<b>0.16</b>	<b>0.57</b>	<b>0.7</b>	<b>0.76</b>	<b>0.79</b>	<b>0.79</b>	<b>0.16</b>	<b>0.53</b>	<b>0.63</b>	<b>0.67</b>	<b>0.69</b>	<b>0.69</b>
	2e-5	0.02	0.34	0.5	0.57	0.59	0.6	0.02	0.33	0.47	0.53	0.55	0.55

batch matrices, facilitating representation in matrix form. When employing aggregation functions, it's essential to ensure that the aggregation process remains unaffected by padding, relying solely on the actual instance data. This consideration can be easily overlooked, such as when computing the mean of the matrix, where the division should be by the effective non-zero rows; not the total number of rows.

We can easily show that the proposed embedding aggregation method is "padding-independent." Let  $M \in \mathbb{R}^{m \times n}$  be a matrix of dimensions  $m$  by  $n$ , and  $Z \in \mathbb{R}^{k \times n}$  denote a zero-padding matrix with matching column dimensions but differing row dimensions. Let  $P \in \mathbb{R}^{(m+k) \times n}$  represent the resulting padding matrix:  $P = \begin{bmatrix} M \\ Z \end{bmatrix}$ .

Then, it is evident that  $P^T P = M^T M$ , but  $P$  is not equal to  $M$ . Also  $\mathbb{1}^T P = \mathbb{1}^T M$ ; for  $\mathbb{1}$  being a corresponding sized one vector. Since  $M$  has no padding and remains independent of  $Z$ , any transformations based on  $P^T P$  and  $\mathbb{1}^T P$  are also unaffected by  $Z$  and, hence, "padding-independent"; and so a valid post-aggregation operation.

**Take-Away**

- We used our proposed unsupervised deep clustering and embedding technique to find similarities among Rebrickable® objects based solely on their compositional features. We found visually similar instances in a close embedding space; even without (explicitly) considering visual features.
- We analyzed supervised classification with strict supervision and semi-supervised tasks using AEs to preserve identity. The results differed depending on whether we used Kernel Embeddings (KEs) and/or Residual Connections (RCs). RCs worked well in pure classification, while a combination of RCs and KEs was superior in semi-supervised tasks with identity/information preservation.
- We conducted a comparative analysis between a brute-force Bert model-based approach and our proposed HDB architecture; for different model sizes (i.e. amounts of learnable parameters); on a classification task of 50 labels. Particularly at comparable model sizes, our proposed HDB solution clearly performed much better, and further, even achieved similar performance with much fewer parameters.
- When training on batches of graph instances, specialized padding aggregations are necessary. Our proposed aggregation function is inherently *padding-independent*.

**Keywords**

Clustering • Deep Embeddings • Padding-Independent • Graph Instances

## 9.5 Manifold Clustering & Embedding

To describe and justify relationships between data, the underlying distribution of data is commonly based on the following assumptions (Learning, 2006):

- *Smoothness*: Similar inputs should yield similar model outputs.
- *Continuity*: Close points are more likely to share the same label
- *Manifolds*: Data resides on latent manifolds of (much) smaller dimensions.
- *Clustering*: Data tends to concentrate into different clusters. Points inside a cluster are more likely to have the same label.

Recall, as mentioned in Chapter 6.6, "Manifold Clustering & Embedding" aims to learn a vector representation for each manifold and assign each data point to a manifold. Recently, Li et al., 2022 argued that two key elements are needed for performing manifold clustering with NNs: a domain-specific constraint for manifold identification and a learning method for embedding manifolds into linear subspaces in the feature space. The *Neural Manifold Clustering and Embedding* (NMCE) integrated these concepts through data augmentation and the principles of *Maximal Coding Rate Reduction* (MCR<sup>2</sup>)(Yu et al., 2020), a technique for learning diverse

and discriminative representations. NMCE adhered to the principles that the clustering and representation should respect domain-specific constraints, that Manifold embeddings should not collapse, and that identified manifold embeddings should be linearized and separated. The NMCE objective loss for data  $X$  was overall as follows:

$$L_{NMCE}(X) := \underbrace{\lambda_0 MCR^2(X)}_{\text{learning algorithm}} + \underbrace{\lambda_1 D(X)}_{\text{identity constraint}} \quad (9.5)$$

where  $\lambda_{0,1}$  are weighting parameters. We bring this again to mind as it has a nice resemblance to our methodology (as mentioned in Chapter 7.7); where our forward-pass  $f(x)$  can be equivalently expressed as:

$$Loss(X) := \underbrace{\lambda_0 \text{Classification}(X)}_{\text{learning algorithm}} + \underbrace{\lambda_1 \text{Auto-Encoder}(X)}_{\text{identity constraint}} \quad (9.6)$$

We also mentioned the use of multiple different loss functions (Chapter 7.5.1), and so, we have  $\mathbf{L} := \{\mathcal{L}_{rec}, \mathcal{L}_{comp}, \mathcal{L}_{sim}, \mathcal{L}_{con}\}$ . Hence, in our case, this changes to:

$$\mathcal{L}_{total} := \underbrace{\lambda_{clas} \mathcal{L}_{clas} + \lambda_{con} \mathcal{L}_{con}}_{\text{learning algorithm}} + \underbrace{\lambda_{rec} \mathcal{L}_{rec} + \lambda_{sim} \mathcal{L}_{sim} + \lambda_{comp} \mathcal{L}_{comp}}_{\text{identity constraint}} \quad (9.7)$$

So  $\mathcal{L}_{clas}, \mathcal{L}_{con}$  are accordingly learning algorithms, and  $\mathcal{L}_{rec}, \mathcal{L}_{sim}, \mathcal{L}_{comp}$  respectively identity constraints.

### Take-Away

To achieve effective manifold clustering with neural networks, two general essential ingredients are required: a domain-specific **constraint that ensures manifold identification and a learning algorithm for embedding each manifold** to a linear subspace in the feature space (Li et al., 2022).

### Keywords

Manifold Clustering • Domain-Specific Constraint • Learning Algorithm

#### 9.5.1 Preservation of Identifying Structures

Building on the concepts of Li et al., 2022 regarding identity-preserving constraints and manifold clustering, we show that combining an unsupervised reconstruction loss of an AE with a supervised classification loss can preserve the data's (identity) structure whilst also effectively separating (i.e. classifying) the data into separate clusters. In particular, we show (Figures 9.6, 9.7) that relying solely on an AE loss does not effectively produce distinct and separable clusters. Conversely, utilizing only a classification loss, while now capable of distinguishing and separating clusters, fails to preserve and encode distinctive (i.e. identifying) structures; this occurs because all sub-groups within a label are treated equally, causing instances within clusters to become mixed and disorganized; losing so their "identity". However, combining an AE loss with a classification loss allows optimizing for both: separating data clusters based on labels and preserving the intrinsic structural identity. This ultimately leads to more enhanced and meaningful vector representations.

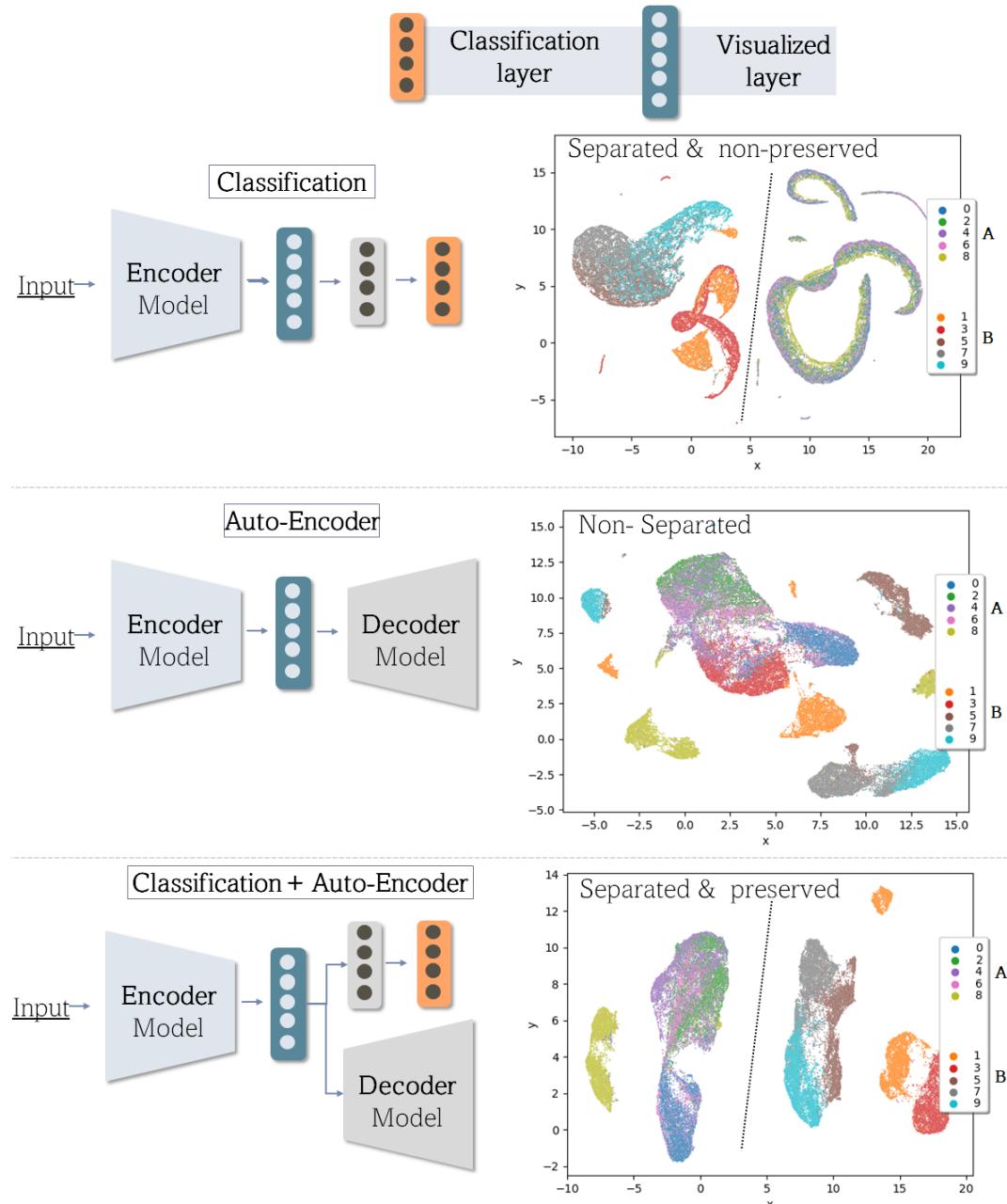


FIGURE 9.6: Projection of latent representations for *FMNIST* using different architectures. Classification labels were obtained by splitting the original ten label indices into two classification groups A & B (based on whether the label indices are even or odd).

It is important to note, however, that the quality of the preservation of identifying structures in this context is heavily reliant on the AE. In particular, if the model is unable to properly encode or decode the underlying data (for example, if the model is too weak given the data's underlying complexity or distribution), the model's determined manifolds may not be representative of identifying patterns and may fail to encode this information into the embeddings. In other words, the importance of choosing the right AE model for a given data collection should not be underestimated. Figure 9.7 shows cases where the AE fails to interpret the data as intended.

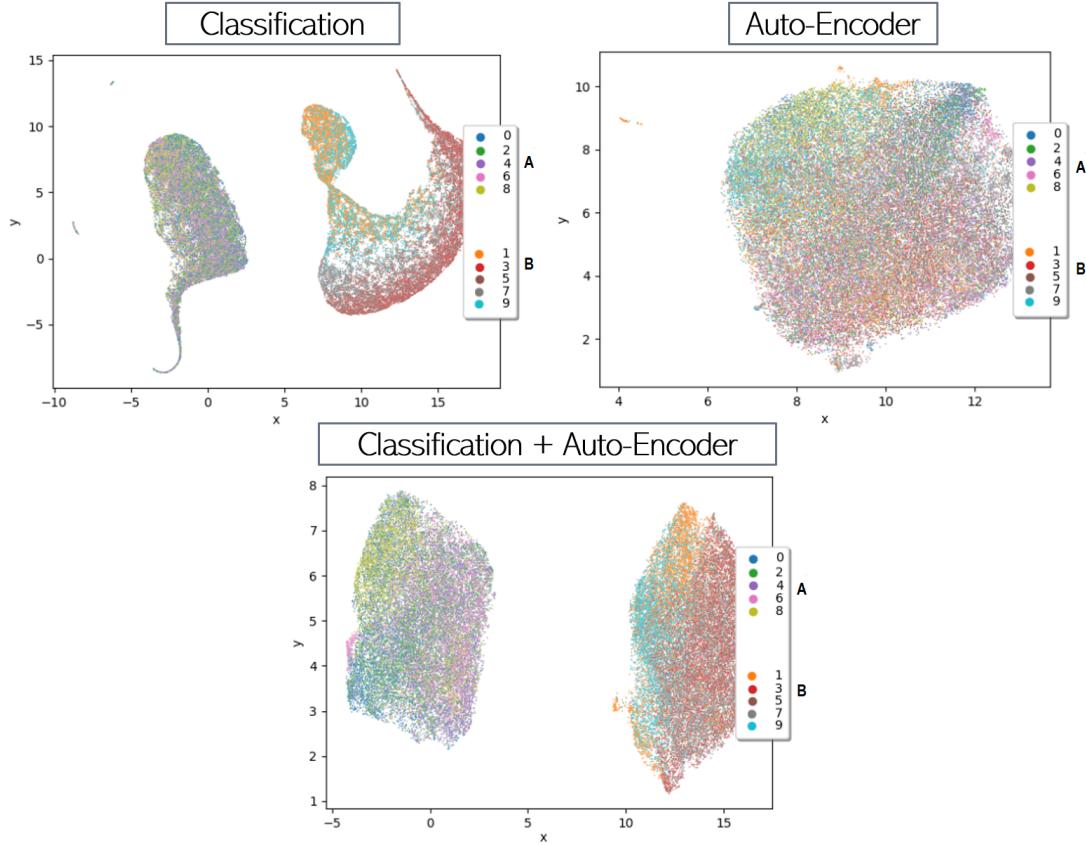


FIGURE 9.7: An example of too-weak AEs that fail to effectively interpret the underlying data and preserve structural identity as intended (finding and preserving sub-labels).

At first, the cluster(s) appear(s) to be distributed randomly with no visible coherence, pattern, or grouping. However, this does not (necessarily) mean that the AE is completely unable to encode (and decode) the data, or that the representations are of absolutely no quality. The embedding vectors may still contain (in some sense) semantic or logical information, just not in a way that efficiently groups, identifies, or showcases items with equivalent ground sub-labels together; i.e. not in the way it was intended. For example, we could have wished for correctly classifying over coarse labels, yet still maintain and not destroy sub-label identities; but a model that is too weak might only succeed in correctly classifying data and fail to preserve identifying distinctions among (latent) groups of sub-labels.

Thus, by having a means of quality measurement for the identity-preservation, we can detect whether the model is too weak or not (the same applies to the classification loss). In the above scenario, the AE serves as this metric; but in general, we can e.g. also use the metrics/losses proposed in Chapter 7.7.

### Take-Away

We can use a multi-loss approach employing an Auto-Encoder (AE) for preserving identity and a classification task as a learning algorithm to produce robust latent vector representations. The AE component maintains identity consistency, while the classification task guarantees a clear separation of labels. Solely relying on either AE losses or classification has drawbacks.

**Keywords**

Multi-Loss • Auto-Encoder • Classification • Embedding Representations

### Taylor Approximation Error During Training

Figure 9.8 below shows the average classification accuracy and the corresponding normalized Taylor approximation error (TE) across five iterations on different data sets. The evaluation encompasses three architectural modules: Classification (CL), Auto-Encoder (AE), and Classification+Auto-Encoder (CL+AE). To introduce coarse labels, the original ten labels were paired (modified) into a total of five groups (i.e. coarse labels), merging pairwise two original labels under a single new one. While some individual differences were found in the results of all the modified data sets, the collective overall empirical observations reveal a number of interesting insights and implications; some of which are not trivially intuitive at first; primarily: (1) discrepancies in train/test error scores, (2) rate of change in the loss error, and (3) variance in the approximation error.

**Train/Test error score discrepancies:** AE showed almost no difference in TE between test and training sets, i.e. the average error curves were nearly identical, CL+TA showed a moderate difference, and CL showed a significantly large difference between training and test error. When the train and test accuracy scores diverged, so did the train and test for TA. This, we believe, may be due to overfitting. Overfitting is easier on a classification task than on an auto-encoder regression task. When overfitting occurs, the TA approximation degrades. On MNIST, CL+AE did not show any divergence in the test and train curves; but had the highest error rate.

**Rate of change:** CL had the strongest rate of decline in the train-TE. In FMNIST (and to a lesser extent in EMNIST), the autoencoder (AE) initially decreased rapidly but then started to increase after a few training epochs. This means that while CL+AE and CL showed a decline in TE towards the end of training, AE instead showed an increase. If we interpret a high TE as indicating that similar input data result in more unpredictable output differences, we can infer that the classification layer maintains predictability and robustness throughout training, whereas the AE becomes less robust (though more effective due to reduced AE error). Even if two input vectors are close, they can still be on different manifolds and thus end up further apart in the output. However, on the MNIST and CIFAR10 data set, this reverse incline did not occur, which could indicate that the manifolds were encoded further apart or that the manifolds were not representative enough so that (numeric) details did not matter. Interestingly, in MNIST, the test error on CL reduced (rather than increasing as in the other data sets); and the training error on CL+AE increased (instead of decreasing).

**Variance in the error:** Overall, throughout the training process (i.e. epochs), the error variance of the curves remained quite stable for all data sets. However, towards the end of training, the error variance of CL decreased somewhat. Additionally, it appeared that the variance of AE was in correlation with the complexity of the underlying data set. Specifically, it exhibited the least variation on CIFAR10 (the most complex/challenging data set) and the greatest variance on MNIST (the easiest).

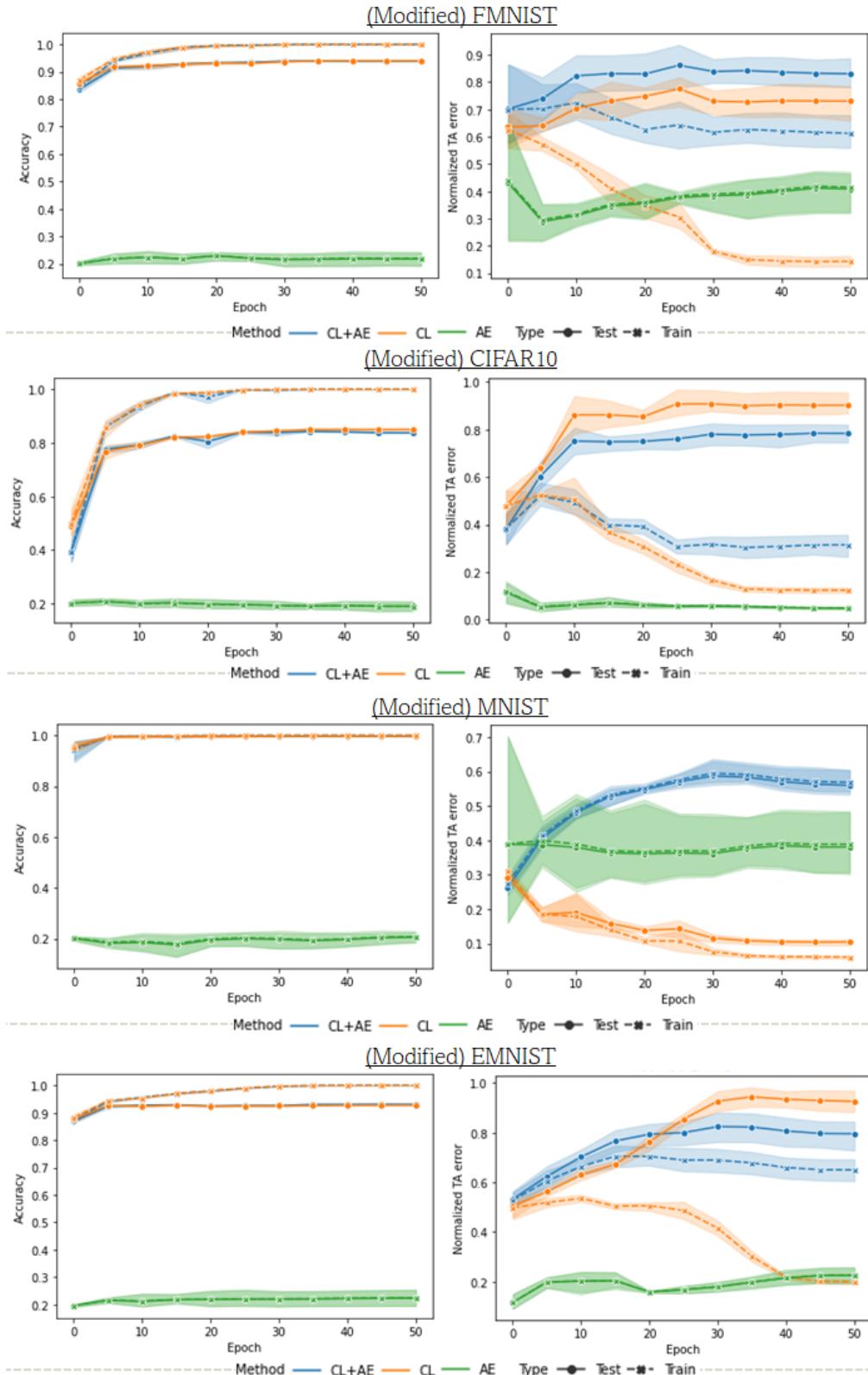


FIGURE 9.8: Accuracy and normalized Taylor approximation error during training on three different architectures.

### Take-Away

Experimentation on the TA error revealed empirical differences in (1) train/test error scores, (2) loss error rate of change, and (3) error variance.

### Keywords

Taylor Approximation Error • Loss Error • Rate of Change • Error Variance

#### 9.5.2 Taylor Approximated Clustering

First, we show that clustering using the TA error over manifolds (i.e. model landscapes) with affinity propagation (AP) can yield more interesting clusters compared to conventional Euclidean-based clustering using HDBSCAN, by visualizing the cluster assignment of an evenly spaced grid field over a given (synthetic) loss landscape, i.e. manifold (Figure 9.9):

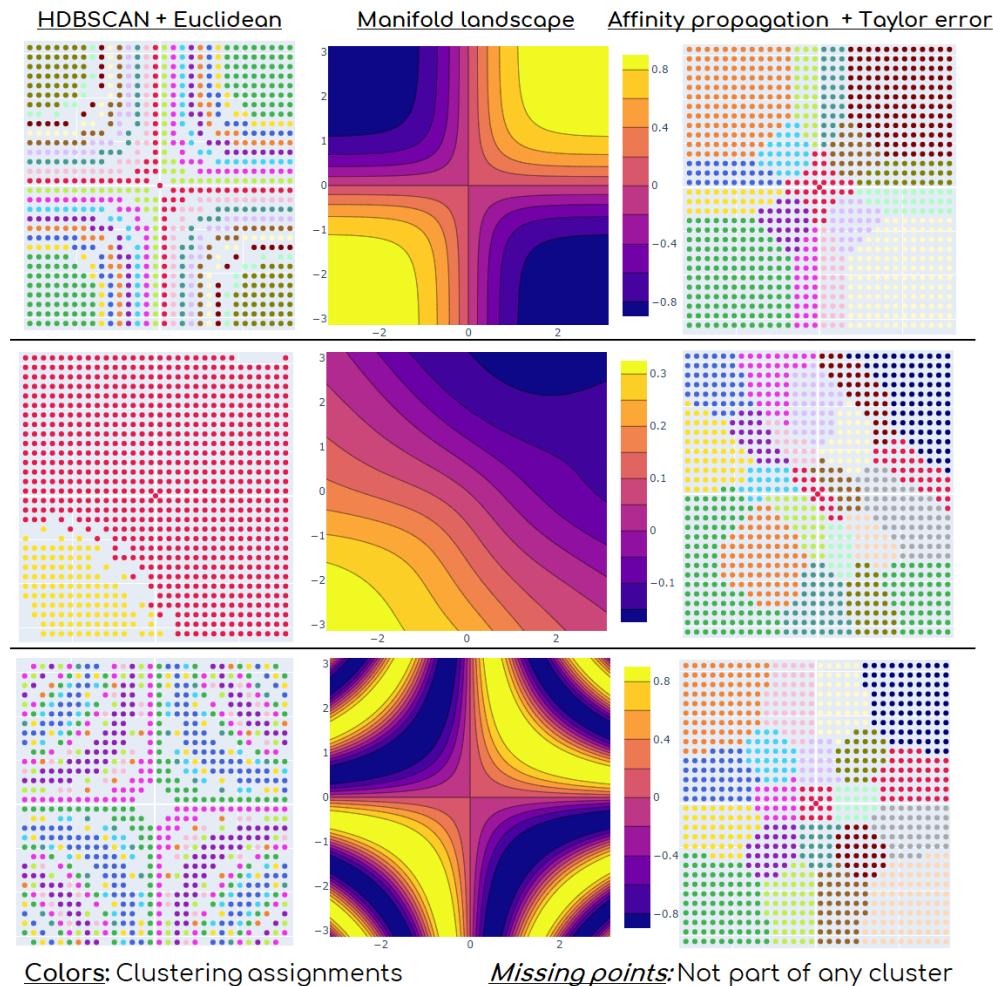


FIGURE 9.9: Comparison between HDBSCAN clustering on the Euclidean distance vs. AP using the TA error.

It is worth noting that the specific and final clustering results - in particular, the size of (merged) clusters - are heavily dependent on the clustering algorithm's hyper-parameter selection; yet there are still clear evident differences between the detected patterns of both methods; even using standard default parameters.

### 9.5.3 Non-Smoothness of DNN Landscapes and Taylor Approximation

The (loss-) landscape of severely over-parameterized DNNs, however, is typically *not* smooth at all and has multiple local and global minima; as can also be visualized using projections (Li et al., 2018a). Minimizing the non-smoothness of network landscapes can lead to increased model effectiveness, especially for sharp loss landscapes (Foret et al., 2020). In our context, training a network to reduce the TA has two intended goals: (1) mitigate landscape irregularity and sharpness by modifying the landscape so that equivalently labeled points may be approximated accurately (potentially improving model efficacy), (2) enable better subsequent manifold clustering since the TA is then more reliable and exact.

We will now highlight and analyze an interesting occurring phenomenon:

**Phenomenon 1** *The TA error of DNNs on training data often worsens (but may also improve) initially during the training process. However, with further iterations, it often tends to gradually converge back to a level similar to the error of the initial TA loss of the untrained DNN. Conversely, the error on test data does generally not improve; and instead worsens until converging towards a maximum error over time.*

Phenomenon 1 supports the following hypotheses and possible implications:

- The DNN’s latent landscape around identically labeled points is *not* smooth, and it worsens as the DNN is trained. The initial untrained DNN’s landscape, on the other hand, has a somewhat smoother (yet not good) TA, i.e. is less unexpectedly abrupt.
- The better the latent representations and generability (i.e. the network loss), the closer similar objects are encoded to other similar objects and the smoother the TA becomes. As a result, until the network has found a first favorable manifold landscape, the TA may first degrade; but then improve.
- Because increasing landscape smoothness is frequently linked with better generalization and model effectiveness, improving the TA could intuitively also possibly result in improved generability and model performance (but only if optimizing the TA smoothed out the landscape appropriately enough).
- It is advantageous to encode similar instances close to one another, such as on a shared local manifold. If the TA on a given manifold is good and smooth, this could indicate a smooth(er) DNN landscape. Ensuring smoothness in (trained) DNNs is challenging due to their inherently non-smooth nature and complex manifolds, often aggravated by overfitting. Rather than directly optimizing the TA, it follows logically that enhancing the smoothness of the DNN’s landscape (and so, that of its manifold) would also lead to improvements in the TA.

#### Take-Away

We examined different clustering methods applied to neural network landscapes and found that combining Taylor approximation error with Affinity Propagation can expose richer patterns compared to traditional approaches. We highlighted the influence of parameter selection on clustering results and explored reducing the Taylor approximation error to improve model performance and ensure consistent clustering; given the intricate nature of non-smooth neural network landscapes.

**Keywords**

Clustering • Neural Network Landscapes • Taylor Approximation Error

## 9.6 Supervised $\alpha$ Max- $B^3$ Clustering Evaluation Metric

**Notice:** Our work and findings on  $\alpha$ Max-B-CUBED (presented in this thesis) have been publicly released on OpenReview-Venue (Guimerà Cuevas and Schmid, n.d.).

We have identified and motivated a potential flaw in the  $B^3$  clustering evaluation metric and described how it can lead to misleading results when dealing with coarse super-labels. This flaw stems from  $B^3$ 's inherent assumption that the ground truth for comparison is accurate and precise, which is not (necessarily) the case when dealing with uncertain or coarse labels. In this section, we will examine different scenarios and analyze different clustering scores, especially considering uncertainty in labels. The term "label uncertainty" here does not refer to that the labels are completely incorrect or wrong; rather, it suggests that they may not accurately denote the exact (sub-)class with complete certainty; i.e., label uncertainty here refers to the uncertainty surrounding *inexact* annotations (Zhou, 2018).

To illustrate, consider the dog label "Chihuahua". While categorizing it as an "animal" is a correct label, a more accurate classification would be a "dog", and an even more precise label would be a "toy dog". Now suppose we have a large data set of many dog images and we want to cluster them without any predefined labels or categories. Since there is no ground truth, it becomes difficult to objectively evaluate the algorithm's performance in terms of labeling accuracy. In this case, one idea may be to incorporate some domain knowledge. For example, we can use breed information as a form of external validation to assess the quality of the dog clusters produced by the algorithm. However, different experts or sources may have varying opinions (labels) on specific dog breeds. For instance, one expert might consider a given dog as a "Retriever," while another might classify it as a "Golden Retriever." This can quickly result in uncertainty regarding (inexact) labels, i.e. coarse labels. Ideally, we would like to have a model that generates distinct clusters for each type of Retriever, rather than grouping all Retrievers into a single cluster. However, if we solely rely on the coarse labels as the absolute (yet inexact) truth, the  $B^3$  will favor the larger cluster over the individual clusters; thus misleading us in assessing which clustering is actually better.

### 9.6.1 Comparison of $\alpha$ Max- $B^3$ and Traditional $B^3$ on Synthetic Data

The proposed  $\alpha$ Max- $B^3$  method was tested on a synthetic data set with five separable classes and compared to the traditional  $B^3$  using *KMeans* for multiple values of  $k$  clusters (Figure 9.10). Both methods performed best for  $k = 5$  and equally for  $k \leq 5$ . However, for  $k > 5$ , the proposed method provided a more robust and fair evaluation, assigning higher scores to sub-clusters and reasonable sub-groups than the traditional  $B^3$ .

### 9.6.2 Imbalanced Data Set

We repeated the above experiment (Section 9.6.1) with *imbalanced* class labels (ratio 1:2:4:8:16) to assess the metric evaluation quality of  $\alpha$ Max- $B^3_\delta$  in an imbalanced data set scenario (Figure 9.11).

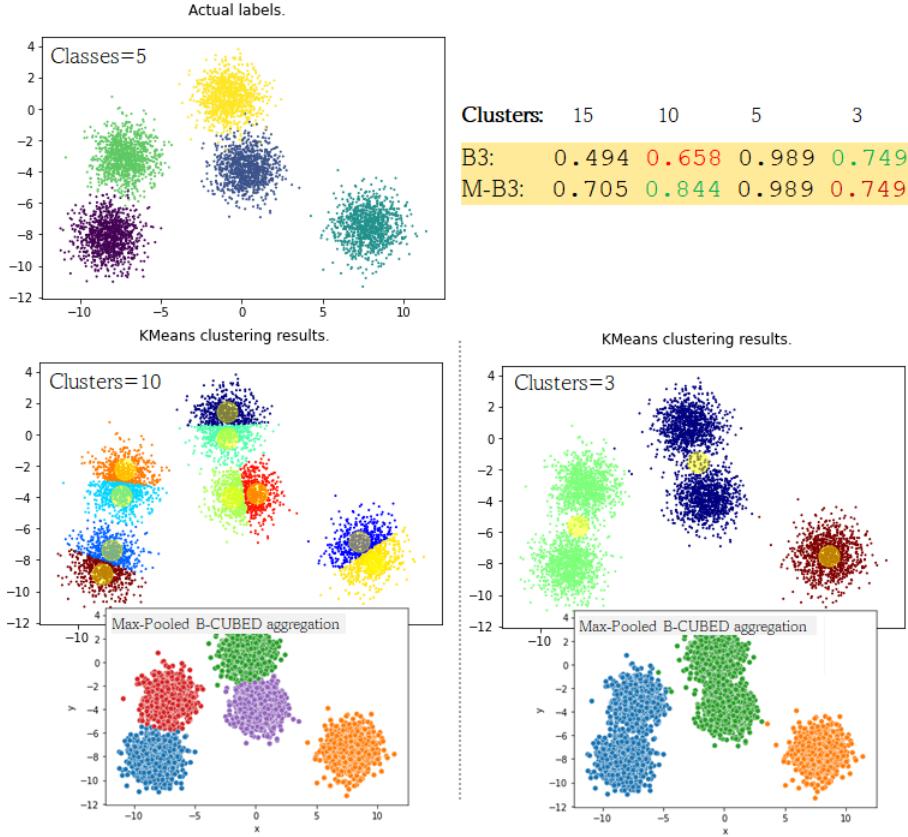


FIGURE 9.10: Multiple clustering assignments are evaluated using clustering scores and depicted visually. The optimal clustering consists of five classes. A finer and purer sub-clustering should have better scores compared to a coarser and less pure clustering. The proposed  $\alpha$ Max- $B^3$  metric, being a variation of the standard  $B^3$ , generates more robust and fair scores. It prioritizes correct extraction of sub-clusters over incorrect super-clusters, while still penalizing non-homogeneous super-clusters when the number of sub-clusters becomes excessive.

$B^3$  and  $\alpha$ Max- $B^3$  have identical scores for  $k \leq 5$  as expected, but the  $\delta$ -balanced variant returned much lower scores, preferring sub-clusters in the most frequent labels and keeping clusters with few points together.

This is consistent with that there is more uncertainty in splitting smaller clusters with few points than larger clusters with many points, because the more points there are, the more likely it is that further sub-groups exist.  $\delta$  was chosen based on the default recommendation (Cui et al., 2019); but other values could have also been used to tweak the balancing.

### 9.6.3 Evaluation of Automatic Uncertainty Determination

We analyzed the automatic uncertainty determination for  $\alpha$  in our method. Our empirical results show that the automatically determined values of  $\alpha$  relate to the actual ground truth uncertainty. We compared our method to the standard  $B^3$  evaluation score and the benchmark scores of the real (unknown) uncertainty value  $\alpha$  to demonstrate this. We used artificial clustering problems with pure and noise groups of cluster assignments, both with and without miss-assignments or outliers. Hence, the actual number of sub-clusters, which is equivalent to the uncertainty, provided

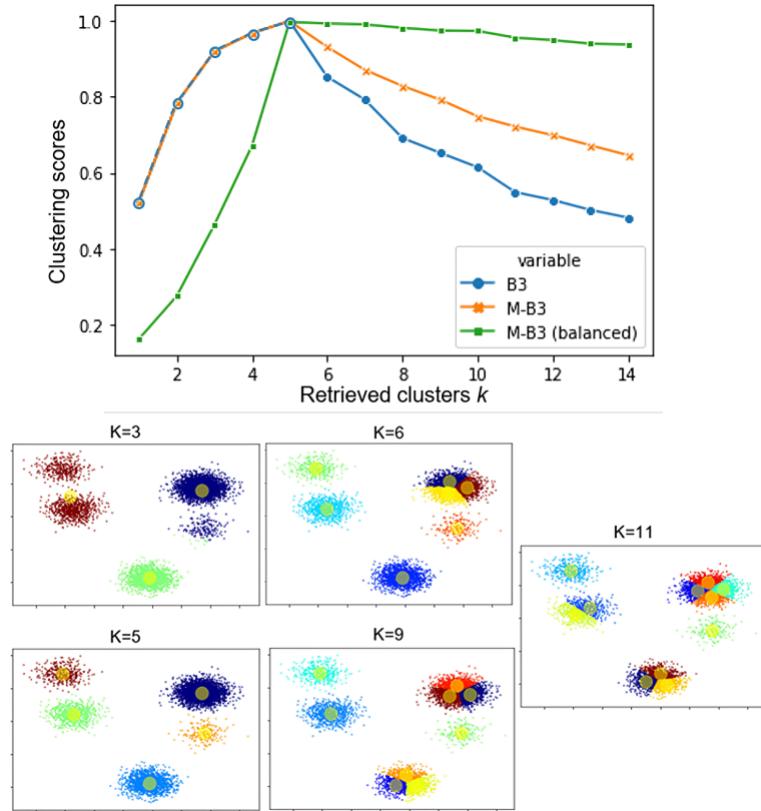


FIGURE 9.11: The  $B^3$ ,  $\alpha$ Max- $B^3$ , and  $\alpha$ Max- $B_\delta^3$  scores on an imbalanced data set of five labels. The performance was evaluated based on different cluster counts  $k$ . On coarse clusters ( $k \leq 5$ ),  $B^3$  and  $\alpha$ Max- $B^3$  perform equally. However,  $\alpha$ Max- $B^3$  performs fairer on finer sub-clusters ( $k \geq 5$ ) than  $B^3$  since it gives more weight to sub-clusters under uncertainty.  $\alpha$ Max- $B_\delta^3$  accounts for class imbalance; which results in differences in  $k < 5$ .

a ground truth benchmark. We set  $k = 10$  classes and evaluated the  $B^3$  score for  $n \in \{1, \dots, 8\}$  sub-clusters per class, requiring a data set of  $k \times 8!$  instances and a total of  $k \times n$  clusters. The ground truth benchmark was thus the uncertainty of  $\alpha = 1 - \frac{1}{n} = \frac{n-1}{n}$ . The results are given in Table 9.2. As can be seen, the scores for the automatically determined values of  $\alpha$  match those of the ground truth uncertainty perfectly (for no noise), and almost perfectly up to about four decimals (with noise). In particular, scores were clearly different from those of the traditional  $B^3$ .

We further performed experiments on data sets exhibiting mixed subgroup uncertainty, defined as the union of two different data sets of equal cardinality, each characterized by distinct subgroup uncertainty. The benchmark was established as the arithmetic mean of the uncertainties of both subgroups; which we denote  $\emptyset \frac{n-1}{n} B^3$ . The results are in Table 9.3. When merging data sets of equal size but with varying numbers of sub-clusters, the scores of the automatic evaluation remained similar, albeit with some notable differences. Increasing the number of samples and the number of class clusters  $k$  only reduced the score difference slightly.

#### 9.6.4 Uncertainty Estimation and Extrapolation

**Notice:** Our work and findings on  $\alpha$ Max-B-CUBED (presented in this thesis) have been publicly released on OpenReview-Venue (Guimerà Cuevas and Schmid, n.d.).

TABLE 9.2: A comparison of scores against ground truth uncertainty.  $n$  denotes the number of real sub-clusters merged under the same visible coarse label. Automatically determining  $\alpha$  yielded here the same scores as using the real uncertainty value (i.e.  $\alpha = \frac{n-1}{n}$ ).

NO NOISE / PURE CLUSTERS			25% RANDOM LABELS			50% RANDOM LABELS					
$n$	B3	$\alpha$ B3	$\frac{n-1}{n}$ B3	$n$	B3	$\alpha$ B3	$\frac{n-1}{n}$ B3	$n$	B3	$\alpha$ B3	$\frac{n-1}{n}$ B3
1	1.0	1.0	1.0	1	0.60657	0.60657	0.60657	1	0.32486	0.32486	0.32486
2	0.66667	0.85714	0.85714	2	0.40330	0.51853	0.51853	2	0.21695	0.27893	0.27892
3	0.5	0.71429	0.71429	3	0.30299	0.43284	0.43284	3	0.16239	0.23198	0.23198
4	0.4	0.60870	0.60870	4	0.24243	0.36892	0.36892	4	0.13023	0.19818	0.19818
5	0.33333	0.52941	0.52941	5	0.20215	0.32106	0.32106	5	0.10871	0.17266	0.17266
6	0.28571	0.46809	0.46809	6	0.17329	0.28390	0.28390	6	0.09242	0.15141	0.15141
7	0.25	0.41935	0.41935	7	0.15143	0.25403	0.25403	7	0.08148	0.13669	0.13669
8	0.22222	0.37975	0.37975	8	0.13491	0.23055	0.23055	8	0.07219	0.12338	0.12338

TABLE 9.3: A comparison of scores from merged data sets of varying levels of uncertainty (sub-cluster tuples) under different degrees of noise, number of instances, and cluster classes.  $(m,n)$  denotes  $m$  coarse labels; each comprising  $n$  sub-clusters.  $\mathcal{O}\frac{n-1}{n}$  is the average uncertainty of both merged data sets.  $c$  denotes the total number of total number of real classes.

[C=10], NO NOISE			[C=10], 25% RANDOM LABELS			[C=10], 50% RANDOM LABELS			
$(m, n)$	B3	$\alpha$ B3	$\mathcal{O}\frac{n-1}{n}$ B3 $(m, n)$	B3	$\alpha$ B3	$\mathcal{O}\frac{n-1}{n}$ B3 $(m, n)$	B3	$\alpha$ B3	$\mathcal{O}\frac{n-1}{n}$ B3
(1,5)	0.5585	0.6731	0.6714 (1,5)	0.3404	0.4116	0.4099 (1,5)	0.1855	0.2253	0.2241
(2,6)	0.3559	0.5118	0.5118 (2,6)	0.2176	0.3126	0.3126 (2,6)	0.1196	0.1723	0.1723
(3,7)	0.2692	0.4164	0.4104 (3,7)	0.1648	0.2546	0.2517 (3,7)	0.0912	0.1412	0.1405
(4,8)	0.2193	0.3532	0.3435 (4,8)	0.1347	0.2167	0.2115 (4,8)	0.0752	0.1211	0.1190

[c=50], NO NOISE			[c=150], NO NOISE			[c=250], NO NOISE			
$(m, n)$	B3	$\alpha$ B3	$\mathcal{O}\frac{n-1}{n}$ B3 $(m, n)$	B3	$\alpha$ B3	$\mathcal{O}\frac{n-1}{n}$ B3 $(m, n)$	B3	$\alpha$ B3	$\mathcal{O}\frac{n-1}{n}$ B3
(1,5)	0.4666	0.6217	0.6198 (1,5)	0.4498	0.6090	0.6073 (1,5)	0.4463	0.6065	0.6047
(2,6)	0.2903	0.4403	0.4403 (2,6)	0.2786	0.4267	0.4267 (2,6)	0.2763	0.4240	0.4240
(3,7)	0.2172	0.3500	0.3434 (3,7)	0.2081	0.3381	0.3313 (3,7)	0.2063	0.3357	0.3289
(4,8)	0.1758	0.2940	0.2828 (4,8)	0.1682	0.2833	0.2719 (4,8)	0.1667	0.2811	0.2697

We extract the level of uncertainty by measuring the plateau interval and extrapolating the  $\alpha$  values and compare the estimation with the ground truth uncertainty of the data set. We do this, by calculating the expectation of all automatically determined alpha values for all labels (i.e., over all function Plateaus intervals using  $\alpha := \min(p_1, p_2)$ ; see Figure 7.33). The results are given in Table 9.4; for a single and merged data sets respectively (as in Section 9.6.3):

Again, on a consistent data set, the uncertainty estimation was perfect. However, on merged data sets with inconsistent (i.e. different) uncertainties, there were slight differences, but the estimation was nevertheless close.

### 9.6.5 Results on Real-World Data

We evaluated model performance using the real-world data set CIFAR100 (using  $\approx 50k$  images) (Krizhevsky, Hinton, et al., 2009). CIFAR100 consists of 100 classes,

TABLE 9.4: Estimating data uncertainty by extrapolating  $\alpha$  and comparing it to the actual uncertainty. The extrapolation was perfect for consistent data sets with a uniform cluster uncertainty. For inconsistent data uncertainty, e.g. merging two different sets with different uncertainties, the extracted  $\alpha$  was not perfect but approximated the uncertainty well.

CONSISTENT DATA SET								MERGED DATA SETS					
REAL $\alpha$	1	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{7}$	$\frac{1}{8}$	REAL $\alpha$	0.6	0.333	0.238	0.188
EXTR. $\alpha$	1	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{7}$	$\frac{1}{8}$	EXTR. $\alpha$	0.5	0.290	0.218	0.172

which can be further divided into 20 super-classes consisting of five sub-classes each. We considered the sub-classification with an underlying label uncertainty of  $\alpha \approx 1/5$ . Details on the CIFAR100's super-classes can be found in the appendix of Krizhevsky, Hinton, et al., 2009.

Our goal was to assess the evaluation quality of a pre-trained computer vision model ( $VIT_{b-32}$ ; Dosovitskiy et al., 2020), which was fine-tuned for the specific classification task. We analyzed the  $B^3$  scores obtained by comparing the predicted labels against both the coarse (super-class) labels and the true ground-truth labels (Table 9.5). We assessed the scores based on the original set of 100 labels as well as the corresponding super-classes of 20 sub-groups. We found that the  $\alpha \text{Max}B^3$  score on sub-labels was higher and, hence, moderately closer to the real scores; compared to the  $B^3$  score.

TABLE 9.5: Comparison of model evaluation on real and coarse labels for CIFAR100 sub-labels using  $B^3$  and  $\alpha \text{Max}B^3$ -CUBED; both, for automatic estimation of  $\alpha$  and using the real  $\alpha$ .  $VIT_{b-32}$  (Dosovitskiy et al., 2020) was used as base model; at multiple different accuracy levels.

Actual scores on real labels	Scores on coarse labels		
	$B^3$	$\alpha \text{ Max}B^3$	$\frac{4}{5} \text{ Max}B^3$
0.99788	0.33316	0.52913	0.52913
0.88420	0.31305	0.49704	0.49704
0.70064	0.27579	0.43693	0.43659
0.58246	0.24880	0.39388	0.39137

### Take-Away

The  $B^3$  cluster quality metric is affected by imprecise and coarse labels, which can lead to inaccurate evaluation results. To address this problem, a modification can be made that is based on mathematical principles. This involves adapting the  $B^3$  metric so that it can handle uncertain and inexact labels that may be present in data sets; achieved by incorporating special super-set aggregations with appropriate weighting. Such modifications help to make the evaluation process more reliable and fair.

**Keywords**

Coarse Labels • Inexact Label Uncertainty •  $B^3$  Cluster Metric • Evaluation

## 9.7 Cluster Explainability using Natural Language

In this section, we explore cluster explainability and interoperability using aggregations on the latent embedding space, i.e. the network’s manifold. We aim to understand how the choice of aggregation method can affect the resulting aggregation vector’s position and semantic meaning. We start by demonstrating that different centrality pooling methods can lead to different vectors. Though this might seem obvious, by explicitly acknowledging this fact, we can gain a clearer and more comprehensive understanding of manifold aggregations. This also helps to highlight that different pooling methods have unique performance characteristics, each with its own advantages and disadvantages, and therefore different implications.

### 9.7.1 Visual Illustration of Embedding Aggregations

For illustration reasons, pooling was here (for the following illustration only; Figure 9.12) applied over the projected coordinates rather than the complete hidden states to more clearly highlight the geometric variances in the different pooling approaches, as UMAP is not guaranteed to perfectly preserve density (McInnes, Healy, and Melville, 2018) and we wanted to keep the linear relationship throughout comparison (UMAP is non-linear). Our intention here is to emphasize the impact of cluster data dispersion on the concept of central tendency (a point around which data points tend to cluster or distribute).

Figure 9.12 shows a 2D UMAP projection for different groups of image clusters (for synthetic data and COCO data), with the different pooled vectors marked for comparison. Applying pooling to lower-dimensional representations simplifies visualizing the different geometric variations, helping in comparing different pooling methods more easily. It also avoids UMAP’s alteration of data density during dimensionality reduction; i.e., data distortion. Keeping a linear context here ensures a clearer understanding of how each pooling method varies. In the actual proposed method, however, the pooling is computed over the **full latent space**. That is, in practice (and in our proposed method), it is crucial to emphasize that the objective is to aggregate embedding vectors within the actual full latent manifold space, *not* within a space that has reduced dimensions. This is important.

### 9.7.2 Generative Captioning Results of Aggregated Embedding Vectors

Figures 9.13 and 9.14 show examples of cluster descriptions generated using the different pooling methods and approaches. To generate these descriptions, we first passed the pooled representation through the decoder. Then, we used a sampling decoding strategy to create multiple candidate captions. Finally, we selected the caption with the highest BERT score.

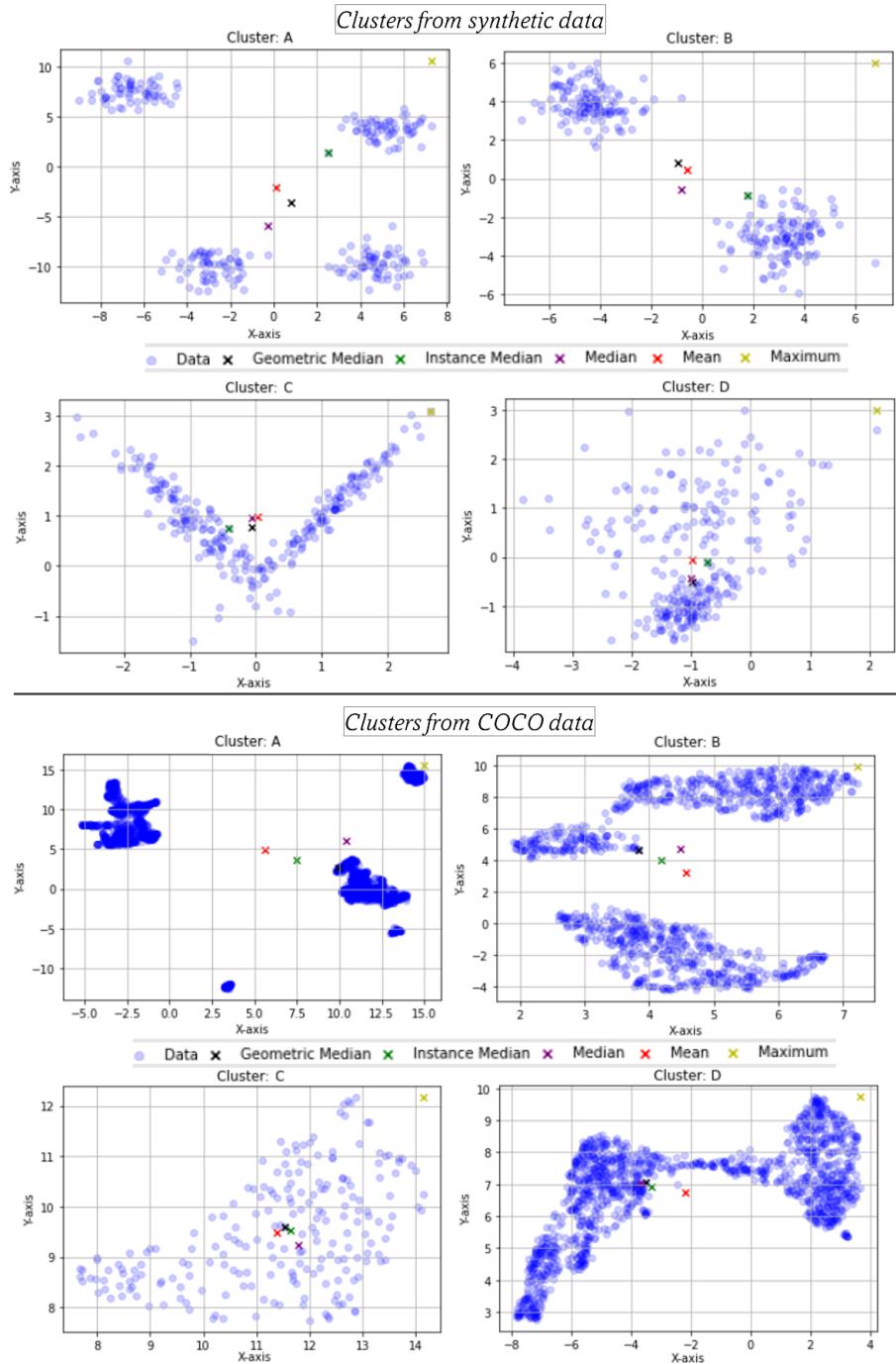


FIGURE 9.12: A comparison of different pooling operations on UMAP projected image cluster groups from synthetic data and the COCO data set. Depicted are: mean-, median-, max-, GM-, and instance GM pooling.

### 9.7.3 Exploring Manifold Aggregation Strategies for Generating Representative Image Cluster Descriptions

We evaluated various pooling methods by analyzing their average text similarity scores on different metrics, with higher scores indicating better performance. For reference, we computed the computationally expensive intra-cluster similarity (the mean similarity of each instance's caption in the cluster compared to all individual

	[0.368]: A hockey player is on the ice. [0.274]: There is a poster of a soccer player. [0.268]: There is an item that says a hockey on the cover. [0.373]: A group of hockey players are playing in a stadium. [0.365]: An icehockey team is getting ready for a game. [0.379]: A group of hockey players are wearing blue jerseys. [0.386]: A hockey player is standing on the ice. [0.386]: A hockey player is standing on the ice. [0.374]: A group of hockey players are gathered in front of the boards.	Max. Intra. Random Max Mean Median GM Medoid CLIP-weigh. CLIP-optim.
	[0.423]: The keyboard has all of the keys. [0.330]: There is a piece of paper with the word Ruby on it. [0.403]: There is a keyboard on the surrounded keys. [0.442]: There is a keyboard with many keys. [0.432]: There is a keyboard with a number of keys. [0.423]: The keyboard has all of the keys. [0.416]: There are keys on a keyboard. [0.434]: There are visible keys on a keyboard. [0.442]: There is a keyboard with many keys.	Max. Intra. Random Max Mean Median GM Medoid CLIP-weigh. CLIP-optim.
	[0.480]: A group of boats are sitting on the water. [0.368]: A collection of underwater memorabilia sitting on [...] [0.230]: A group of some is standing on a bench. [0.503]: Many boats are docked at the large harbor. [0.531]: A large amount of boats are docked along a waterfront. [0.490]: A large number of boats are parked in the sea. [0.516]: Several boats are docked next to the water. [0.486]: An assortment of large and small boats on water next to dock. [0.550]: A large number of boats are docked near the water.	Max. Intra. Random Max Mean Median GM Medoid CLIP-weigh. CLIP-optim.
	[0.440]: A glass cup holding three toothbrushes next to a wall. [0.350]: The antique toothbrush is still in its holder. [0.400]: A tooth brush stands on top of a bed. [0.433]: Three toothbrushes in varying sizes are next to a tube. [0.428]: A bathroom mirror filled with four tooth brushes. [0.422]: A couple of toothbrushes sitting on a bathroom counter top. [0.448]: Two colorful toothbrushes are sitting inside a bathroom mirror. [0.448]: A couple of toothbrushes sitting on a glass shelf. [0.450]: Two toothbrushes are in a glass container next to paste.	Max. Intra. Random Max Mean Median GM Medoid CLIP-weigh. CLIP-optim.

FIGURE 9.13: [Extract 1] A sample of CCs generated from TextCaps data (Sidorov et al., 2020) using diverse pooling techniques; with corresponding BERT-scores. Cluster captions were produced by passing the pooled representation through a decoder, applying a sampling decoding strategy, and choosing the caption with the highest BERT-score. Four clusters are shown, each with a random subset of nine cluster instances.

captions within the same cluster) using brute-force comparisons and retrieved the average (mean intra-cluster) and maximum (max intra-cluster) instance similarity. The results are summarized in Table 9.6, which we further ranked relative to their average performance per decoding strategy (Table 9.7).

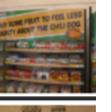
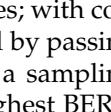
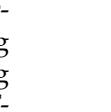
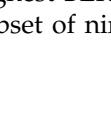
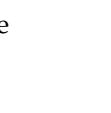
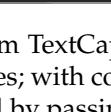
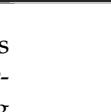
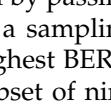
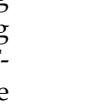
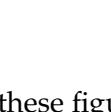
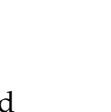
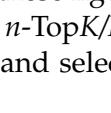
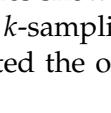
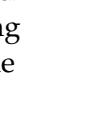
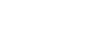
<u>Max. Intra.</u>	[0.379]: There is a candy bar display in the store.			
<i>Random</i>	[0.340]: A person is standing next to a container.			
<i>Max</i>	[0.321]: There is is labeled to written to the store.			
<i>Mean</i>	[0.402]: There are items for sale in the store.			
<i>Median</i>	[0.402]: There are items for sale in the store.			
<i>GM</i>	[0.399]: There is an aisle for beverages in the liquor store.			
<i>Medoid</i>	[0.405]: There is an aisle in the store with candy.			
<i>CLIP-weigh.</i>	[0.401]: There is an aisle for sale in the store.			
<i>CLIP-optim.</i>	[0.411]: There is a store aisle with many items.			
<u>Max. Intra.</u>	[0.424]: There is a box with a label next to it.			
<i>Random</i>	[0.334]: There is a sign above the bikes.			
<i>Max</i>	[0.143]: an to in it.			
<i>Mean</i>	[0.403]: There is a box with a label on it.			
<i>Median</i>	[0.424]: There is a box with a label next to it.			
<i>GM</i>	[0.403]: There is a box with a label next to another box.			
<i>Medoid</i>	[0.399]: There is a box with a logo on it.			
<i>CLIP-weigh.</i>	[0.424]: There is a box with a label next to it.			
<i>CLIP-optim.</i>	[0.424]: There is a box with a label next to it.			
<u>Max. Intra.</u>	[0.456]: A small horse is running around the grass field.			
<i>Random</i>	[0.322]: A brown and white cocked animal standing [...]			
<i>Max</i>	[0.284]: A sitting onA horse with their horse			
<i>Mean</i>	[0.444]: A couple of brown horses [...] on a grass covered field.			
<i>Median</i>	[0.451]: A pair of horses stand on a grassy field.			
<i>GM</i>	[0.441]: A couple of horses [...] on top of a dry grass field			
<i>Medoid</i>	[0.447]: A pair of horses walking through a field of grass.			
<i>CLIP-weigh.</i>	[0.465]: A couple of horses standing in a grassy field.			
<i>CLIP-optim.</i>	[0.470]: A couple of horses are standing in a grassy field.			
<u>Max. Intra.</u>	[0.480]: Three glasses of wine sitting on a wooden table.			
<i>Random</i>	[0.414]: A set of three wine glasses next to a large bird.			
<i>Max</i>	[0.359]: A man sitting behind a wine on a table			
<i>Mean</i>	[0.480]: A couple of wine glasses sitting on top of a table			
<i>Median</i>	[0.494]: A set of three glasses filled with wine [...] a table			
<i>GM</i>	[0.502]: A couple of glasses filled with wine [...] a table.			
<i>Medoid</i>	[0.432]: A hand holding a glass of wine near a wooden table.			
<i>CLIP-weigh.</i>	[0.492]: A couple of glasses of wine sitting on a wooden table.			
<i>CLIP-optim.</i>	[0.487]: A couple of glasses of wine sitting on top of a table.			

FIGURE 9.14: [Extract 2] A sample of CCs generated from TextCaps data (Sidorov et al., 2020) using diverse pooling techniques; with corresponding BERT-scores. Cluster captions were produced by passing the pooled representation through a decoder, applying a sampling decoding strategy, and choosing the caption with the highest BERT-score. Four clusters are shown, each with a random subset of nine cluster instances.

To draw a comparison to the Figures 9.13 & 9.14 of before, these figures showed CCs of different pooling techniques obtained using  $D_{|\varrho}$  with the  $n$ -TopK/P k-sampling decoding strategy  $\varrho$ ; which sampled multiple CC candidates and selected the one

TABLE 9.6: Average text similarity scores for different pooling methods using various metrics with different scales. Higher scores indicate better performance; the average and maximum cluster-wise score using a brute-force search is included as a reference.

TEXTCAPS	BRUTE-FORCE SEARCH		BLEURT	BERT	BART
MEAN. INTRA-CLUSTER			0.3377	0.7910	-5.5917
MAX INTRA-CLUSTER			<u>0.4174</u>	<u>0.8352</u>	<u>-4.6920</u>

METHODS	GREEDY			BEAM-SEARCH			DIVERSE BEAM-SEARCH		
	BLEURT	BERT	BART	BLEURT	BERT	BART	BLEURT	BERT	BART
RANDOM	0.2817	0.7756	-5.7390	0.2580	0.7474	-5.8207	0.2675	0.7520	-5.9917
MAX	0.2468	0.7378	-6.2386	0.2679	0.7538	-6.0829	0.2573	0.7422	-6.3703
MEAN	0.3707	0.8148	-5.1829	0.3491	0.7998	-5.2575	0.3715	0.8096	-5.2399
MEDIAN	0.3728	0.8152	-5.1343	<u>0.3634</u>	0.8009	-5.1803	0.3684	0.8089	-5.2261
GM	0.3762	0.8186	-5.0998	0.3579	0.7976	-5.2128	<u>0.3753</u>	0.8124	<u>-5.1687</u>
MEDOID	0.3681	0.8112	-5.2485	0.3488	0.7939	-5.4726	0.3589	0.8023	-5.4588
CLIP <sub>weighted</sub>	0.3714	0.8154	-5.1779	0.3491	0.7998	-5.2575	0.3701	0.8105	-5.2175
CLIP <sub>optimized</sub>	<u>0.3836</u>	<u>0.8221</u>	<u>-5.0928</u>	0.3621	<u>0.8035</u>	<u>-5.1991</u>	0.3750	<u>0.8141</u>	<u>-5.1751</u>

METHODS	MULTINOMIAL SAMPLING			BEAM-SAMPLING			<i>k</i> -SEQUENCE SAMPLING		
	BLEURT	BERT	BART	BLEURT	BERT	BART	BLEURT	BERT	BART
RANDOM	0.2644	0.7554	-6.0537	0.2731	0.7585	-5.6139	0.3379	0.8024	-5.1016
MAX	0.2277	0.7299	-6.7867	0.2582	0.7506	-6.1864	0.3459	0.7982	-5.4607
MEAN	<u>0.3484</u>	0.7979	<u>-5.4875</u>	0.3596	0.8043	-5.1944	0.4195	0.8361	-4.6697
MEDIAN	0.3341	0.7898	-5.5030	0.3684	0.8112	-5.0820	0.4191	0.8349	-4.6834
GM	0.3445	0.7937	-5.5289	<u>0.3704</u>	0.8099	<u>-5.0490</u>	0.4179	0.8364	-4.6710
MEDOID	0.3398	0.7966	-5.5276	0.3630	0.8048	-5.3662	0.4167	0.8358	-4.6885
CLIP <sub>weighted</sub>	0.3424	0.7927	-5.5930	0.3647	<u>0.8126</u>	-5.0850	0.4212	0.8369	<u>-4.6541</u>
CLIP <sub>optimized</sub>	0.3456	<u>0.8002</u>	-5.4931	0.3676	0.8091	-5.0781	<u>0.4221</u>	<u>0.8371</u>	-4.6618

COCO-2017	BRUTE-FORCE SEARCH		BLEURT	BERT	BART
MEAN. INTRA-CLUSTER			0.3638	0.8092	-5.531
MAX INTRA-CLUSTER			<u>0.4424</u>	<u>0.8446</u>	<u>-4.755</u>

METHODS	GREEDY			BEAM-SEARCH			DIVERSE BEAM-SEARCH		
	BLEURT	BERT	BART	BLEURT	BERT	BART	BLEURT	BERT	BART
RANDOM	0.2875	0.7684	-5.9931	0.2677	0.7616	-6.1271	0.2667	0.7599	-6.2352
MAX	0.2077	0.7038	-7.0332	0.2187	0.7087	-6.8117	0.2134	0.7082	-6.9270
MEAN	<u>0.4099</u>	<u>0.8329</u>	-5.0400	0.4051	0.8286	-5.0974	0.4051	0.8291	-5.1381
MEDIAN	0.4056	0.8317	-5.0859	0.4028	0.8265	-5.1323	0.3992	0.8280	-5.1665
GM	0.4077	0.8322	-5.0719	0.4032	0.8263	-5.1427	0.4018	0.8286	-5.1692
MEDOID	0.3925	0.8280	-5.1936	<u>0.4070</u>	<u>0.8295</u>	<u>-5.1516</u>	0.3963	0.8278	-5.1852
CLIP <sub>weighted</sub>	0.4098	0.8328	<u>-5.0374</u>	0.4044	0.8284	-5.1164	<u>0.4061</u>	0.8294	-5.1392
CLIP <sub>optimized</sub>	0.4028	0.8324	-5.0567	0.4034	<u>0.8295</u>	<u>-5.0909</u>	0.4007	<u>0.8302</u>	<u>-5.1217</u>

METHODS	MULTINOMIAL SAMPLING			BEAM-SAMPLING			<i>k</i> -SEQUENCE SAMPLING		
	BLEURT	BERT	BART	BLEURT	BERT	BART	BLEURT	BERT	BART
RANDOM	0.2661	0.7535	-6.2851	0.2858	0.7629	-5.9524	0.3314	0.7886	-5.5360
MAX	0.1968	0.7260	-7.2926	0.2130	0.7146	-6.9062	0.3064	0.7846	-6.0822
MEAN	0.3734	0.8073	-5.4669	0.4098	0.8322	-5.0357	0.4408	0.8440	-4.7514
MEDIAN	<u>0.3843</u>	<u>0.8194</u>	<u>-5.3714</u>	<u>0.4179</u>	<u>0.8333</u>	<u>-5.0145</u>	0.4397	0.8431	-4.7562
GM	0.3644	0.8131	-5.4871	0.4086	0.8300	-5.1024	0.4402	0.8436	-4.7652
MEDOID	0.3686	0.8102	-5.4709	0.4038	0.8299	-5.1093	0.4336	0.8417	-4.7897
CLIP <sub>weighted</sub>	0.3751	0.8150	-5.4234	0.4121	0.8313	-5.0570	<u>0.4414</u>	<u>0.8445</u>	<u>-4.7388</u>
CLIP <sub>optimized</sub>	0.3780	0.8177	<u>-5.3548</u>	0.4111	0.8321	<u>-5.0219</u>	0.4379	0.8436	-4.7509

with the highest score.

The intra-cluster brute-force search is computationally heavy as it requires comparing the similarity (e.g. the BERT-score) of each caption to all other captions, and each cluster instance may come with multiple captions. In our generative model, we mainly either produced a single sequence (e.g. when using greedy or beam-search), or for the population-based approach of *k*-sequences, *k* was typically much smaller than the number of instances in the clusters. If we had set *k* equal to the size of the

TABLE 9.7: The average relative rank (smaller is better) of the scores among all methods per decoding strategy.

DATA SET	$\varnothing$ SCORE RANK PER METHOD							
	RANDOM	MAX	MEAN	MEDIAN	GM	MEDOID	CLIP <sub>weighted</sub>	CLIP <sub>optimized</sub>
TEXTCAPS	7.167	7.833	3.833	3.611	2.778	5.500	3.556	1.722
COCO-2017	7.000	8.000	2.667	3.500	4.500	5.333	2.389	2.611
OVERALL	7.084	7.917	3.250	3.556	3.639	5.417	2.973	2.166

cluster captions, both performance and evaluation would have been significantly more computationally intensive; but CC quality would then have increased as well.

### Take-Away

In a cluster, items are grouped together because they are somehow close in proximity. By combining the representation of these instances, we can generate a textual description that is meant to represent the cluster as a whole. The method of averaging used (e.g. a simple mean, median, or a more complex approach) affects the quality of the generated outputs. To further enhance the final textual descriptions, we can sample multiple candidate texts (based on a single cluster representation) and rank them using a quality metric or heuristic. Finally, we select the most matching one with the highest score.

### Keywords

Clustering • Proximity • Representation Averaging • Quality Selection

## 9.8 Normalization of Heterogeneous Feature distributions with Adaptive Tanh-Estimators

**Please note:** This contribution has been accepted for publication, after peer review in Guimerà Cuevas and Schmid, 2024b. The Version of Record is available online.

### 9.8.1 Classification With Neural Networks

We evaluated the effectiveness of NNs in classifying synthetic data (Figure 9.15), computer vision toy data (Figure 9.16), and real-world data (Figure 9.17); for different spread values. The features of synthetic and real-world data were normalized individually. Pixel-normalization for vision sets was on the gray-scale channel. The goal was to examine the impact of FS on the training for different tasks. Specifically, we show that inadequate spread initialization in TEs can harm training convergence; which highlights the importance of choosing a proper normalization spread value.

Training converged much faster with feature-wise ideal spread values  $\hat{\alpha}$  than with a fixed global default spread value for all features of  $\alpha = 0.01$ , as expected. On synthetic tabular data (points drawn from linear Gaussian combinations), an  $\alpha = 1$  exhibited a similar training convergence as  $\hat{\alpha}$ , but the value range was too large. Notice, how  $\alpha = 1$  is equivalent to directly applying tanh to the standardization. Conversely, using a global default spread of  $\alpha = 0.01$  achieved the worst performance on the synthetic data set, and considerably constrained the effective

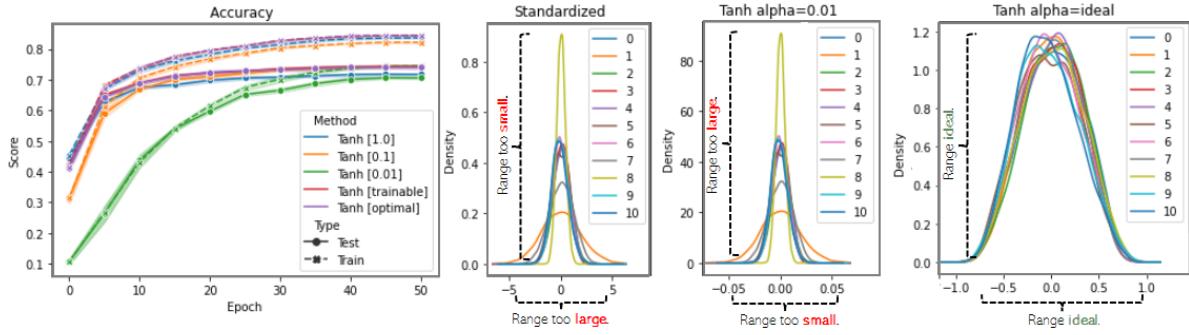


FIGURE 9.15: Comparing normalized KDEs and model performance on synthetic data. Training accuracy is depicted in the left figure; the others show the KDEs of the normalized features. A default spread of  $\alpha = 0.01$  showed the worst convergence.  $\alpha$  resulted in the best probability mass distribution.

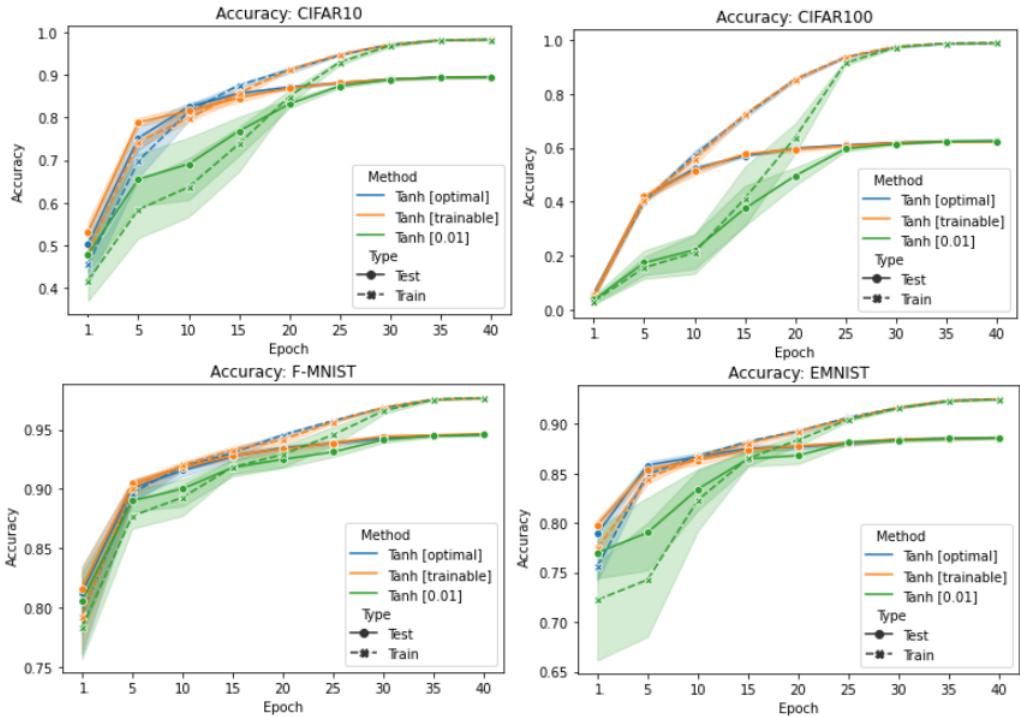


FIGURE 9.16: Training performance of ideal, trainable, and fixed spread values on four different toy computer vision data sets after tanh-normalizing the gray-scale pixel value distribution.

feature range. Overall, the probability mass of the KDE of the normalization for  $\alpha$  produced the best distribution, with all features having similar peak densities and being Gaussian-like inside  $[-1, 1]$ . On the computer vision sets,  $\alpha$  demonstrated substantially less performance variance between independent runs. Here, although the ideal spread values  $\alpha$  clearly improved training convergence speed, the final scores towards the end of the training were about the same for all  $\alpha$  values. This suggests that a good choice of  $\alpha$  strongly contributes to a better training start and reduces or eliminates the need to adjust training weights to poorly chosen feature distribution. Furthermore, recall that, while features are initially normalized independently, back-propagation (optionally) allows us to alter each spread value in relation to the other features. Yet, the performance of  $h_\alpha$  and  $h_\gamma$  for a trainable  $\gamma := \tau\alpha$  was nearly equal

and did not show any noticeable performance gains. The motivation here was that, since  $\alpha$  is calculated based on a particular target distribution, the optimal FS for a given task might require a different target distribution, e.g. one that is not Gaussian. By allowing the parameter  $\tau$  to adjust (i.e. be learned), the spread value can further be adapted towards the "optimal" spread for a given task. Lastly, we analyzed the classification performance of different spread values on the larger real-world CDC Diabetes Health Indicators data set (Figure 9.17). Again, the adaptive normalization outperformed the use of a fixed global spread value in terms of convergence speed. But as before, fine-tuning  $\tau\alpha$  did not substantially improve performance.

Overall, our primary finding was that slow convergence rates in tanh normalizers were primarily due to poorly dispersed density masses in the scaled feature distributions. Inadequate spread values resulted in bad density dispersion and forced initial model weights to adjust to the input domain, thereby slowing down convergence speed. Thus, a fixed global spread value is sub-optimal; but ideal spreads are easily determined.

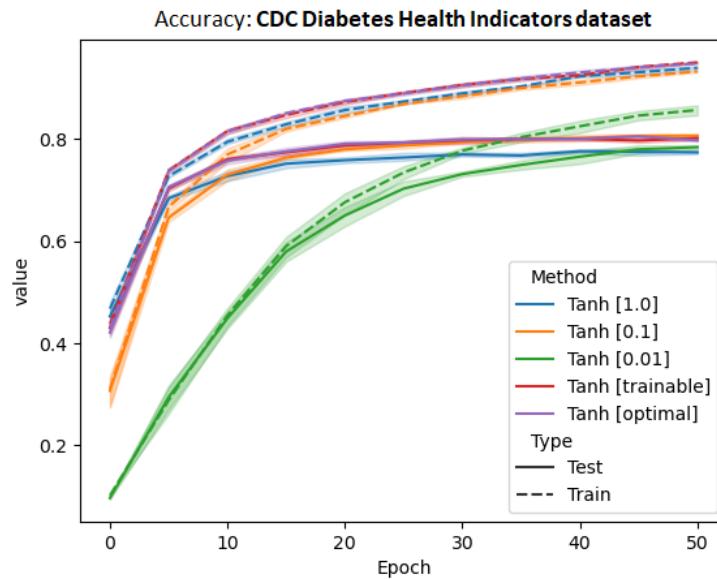


FIGURE 9.17: Classification scores on the CDC Diabetes Health Indicators data set for different spread values.

### 9.8.2 Analytical Optimization

Although the probability mass of a Gaussian is distributed symmetrically following a tanh transformation, a default spread of  $\alpha = 1$  is not suitable since the output probability density places too much weight on the tails. The transformation might be even worse for other standardized (but non-Gaussian) inputs. Instead of being scattered at the extremes, the bulk of the output probability mass should be spread around the center. That is, having the majority of the mass in the middle and just a small amount (e.g. outliers) squeezed on the tails. The spread parameter of the tanh mapping can be changed to tweak this trade-off. For example, if  $\alpha$  is excessively large, the absolute value of practically all features will be extremely close to one. Similarly, if  $\alpha$  is too low, the majority of values will be close to zero. Both situations are undesirable because they contradict the purpose of FS. As a result, the target distribution helps to disperse the probability mass appropriately. Figure 9.18 below

shows the kernel density estimate<sup>1</sup> (KDE) of a truncated min-max normalized target Gaussian with  $q = 0.001$  following the tanh-transformation, i.e. the KDE of  $h_\alpha$ , for various spread values  $\alpha$ . In particular, for data  $X$ , the kernel estimate analyzed is  $\text{KDE}(h_\alpha(X))$ . Notice that  $\text{KDE}(X) \neq h_\alpha(X)$ . The spread value has a strong influence on the KDE of the tanh normalization, with very low spread values strongly squishing the probability mass towards the center.

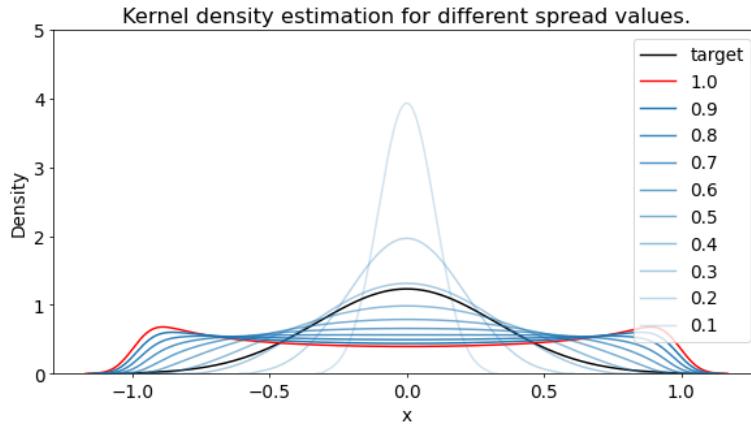


FIGURE 9.18: A truncated standard Gaussian with  $q=0.001$  (black) is compared to the KDE of its tanh-normalization output for various  $\alpha$  spread values (blue). The result for  $\alpha = 1$  is shown in red.

Yet, often, inputs are not normally distributed. Standardization assures a mean and standard deviation of zero and one, respectively, but does not change the underlying (histogram) distribution, i.e. the distribution is relocated and stretched/shrunk, but not "distorted". By minimizing the loss, we merely reduce the mismatch between the output's KDE and a desired target distribution. In other words, we find the spread value  $\alpha$  which results in the lowest possible WD loss after applying the tanh normalization. It is worth noting that even if we apply additional *linear* transformations afterward on the output, such as constraining the range to  $[0,1]$  instead of  $[-1,1]$ , the optimal spread value  $\alpha$  remains the same, but the WD loss may vary. Figure 9.19 below depicts the WD losses of different distributions for different spread values. Empirically, it appears that, for a standard Gaussian as the target, minimizing over the spread value  $\alpha$  may be quasi-convex and feature a unique (global) minimum. The loss itself is, however, not convex. The loss function is also not symmetric, despite point symmetry of tanh; remember, it is non-linear. Hence, adding or subtracting a spread value  $\epsilon \pm \alpha$  for a distance  $\alpha > \epsilon > 0$  has a different impact. E.g., the WD may increase "left" from  $\alpha$  more strongly than "right" of it.

A standard (non-truncated) Gaussian has  $\approx 68.27\%$  of its probability density within a range of  $[-1, 1]$ ,  $\approx 27.18\%$  for  $[-2, -1) \cup (1, 2]$ ;  $\approx 4.28\%$  for  $[-3, -2) \cup (2, 3]$ ; and  $\approx 0.27\%$  elsewhere. For tanh-normalization, the percentages are subject to the spread value. Given a standard normal distributed input, for  $\alpha = 1$  more than two-thirds of the transformed features will have *absolute* values less than 0.762, and more than a quarter will have *absolute* values in  $[0.762, 0.964]$ . Similarly, the default spread value from the literature of 0.01 (Jain, Nandakumar, and Ross, 2005; Latha and Thangasamy, 2011), on the other hand, would here squish more than 99.8%. This raises the question of why such a convention has persisted as this can be considered a terrible scale for feature normalization and may explain why practitioners would resort to different alternative (manually adjusted) spread values. In fact,

<sup>1</sup>An estimation that uses a continuous probability density curve to describe the data distribution.

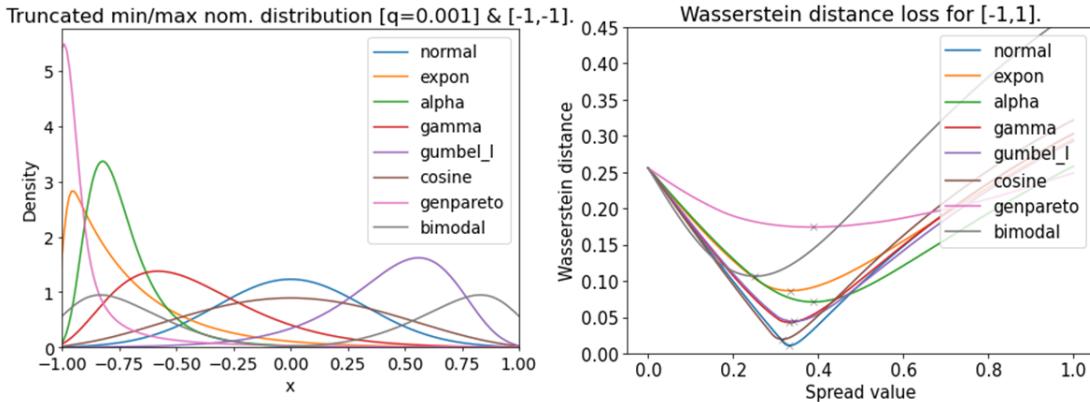


FIGURE 9.19: Wasserstein loss against different spread values for various standardized distributions.

$\alpha = 0.01$  was initially chosen for a transformation using Hampel, rather than for the standardized features in MTN, but manual tweaking and correction were even for Hampel still involved and often necessary (Jain, Nandakumar, and Ross, 2005).

Table 9.8 shows the probability density coverage for various spread values. Entries represent the supremum of the *absolute* output value for the tanh normalization for a certain percentage of the probability density. The target distribution is a standard Gaussian distribution.

TABLE 9.8: Approximate probability density coverage.

$\alpha$	68.2%	95.4%	99.8%	100%
0.01	0.0010	0.0020	0.0030	1
0.10	0.0997	0.1974	0.2913	1
0.25	0.2449	0.4621	0.6351	1
0.50	0.4621	0.7616	0.9051	1
0.75	0.6351	0.9051	0.9780	1
1.00	0.7616	0.9640	0.9951	1

It emphasizes the issue of selecting spreads that are too small, as the maximum absolute values remain extremely low. Too large spread values, however, were not as detrimental, but still far from ideal.

Close to zero, the tanh function is almost linear, with non-linearity more apparent on larger (absolute) values. A very small spread value, yielding a very small effective range, introduces little non-linearity. Consequently, apart from excessively compressing feature values, this minimal non-linearity fails to diminish the impact of outliers or enhance robustness against noise. Weights will have to re-adapt throughout training, but linearity within individual features is largely "preserved". Large spread values, on the other hand, introduce more non-linearity and greater affect the normalization's distribution. If non-linearity is imposed too strongly, potentially assumed linear relationships (e.g. on the weighted sum) may become distorted. Overall, already values  $\leq 0.1$  exhibit a considerable WD divergence, and so the tanh estimator's default spread of 0.01 is a very unfortunate choice here. Table 9.9 contains an overview of spread values and their numerical loss for different ground distributions. The ideal spread value for a Gaussian target distribution was often similar across most distributions; yet not identical.

TABLE 9.9: Ideal spread values.

Distribution	$\approx \alpha$	$\mathcal{L}_W^{q=0.001}$
Normal	0.3328	0.011
Exponential	0.3336	0.087
Alpha	0.3885	0.071
Gamma	0.3326	0.043
Gumbel_L	0.3422	0.044
Cosine	0.3153	0.020
GenPareto	0.3873	0.174
Bimodal	0.2516	0.107

### Take-Away

It is crucial to pre-process feature values when using ML models. Tanh-Normalization is a robust method that can handle outliers well, but it uses a uniform fixed global spread value for all the features during normalization, which is not optimal. One way to improve this is by instead directly calculating the most "ideal" local spread factor for each feature. This helps with training convergence and eliminates the need for manual parameter tuning.

### Keywords

Pre-Procesing • Tanh-Normalization • Outliers • Training Convergence

## 9.9 Model Output Calibration & Novelty Detection

**Please note:** This contribution has been accepted for publication after peer review in Guimerà Cuevas and Schmid, 2024a and has been presented in the conference proceedings.

### 9.9.1 Score Adjustment

In an embedding space, the distances between a point and class centroids often represent the similarities between that point and each class; or its probability of belonging to that class. A class may have "subspaces", e.g. contour lines, along which the distance to the same class remains constant. However, the distances to other classes change (*dark knowledge*). This is visualized in Figure 9.20.

In this context, it is crucial to consider the distances (i.e. mismatches) to other classes for our OOD approach; but a high probability class (low distance to a specific cluster center) can steal away scoring mass from other classes during calibration scoring since the probabilities essentially sum up to one. Thus, it is important to inversely weigh the distances (i.e. scores) if we want to ensure that the scoring mass is distributed appropriately among all classes; recall, we are interested in the average calibration discrepancy, i.e. misalignment, among all classes. For example, the main class may exhibit no mismatch, but all other classes may. Thus, we assign correction weights to these classes in an inverse manner to ensure all classes have a sufficient influence on the final outlier alignment score.

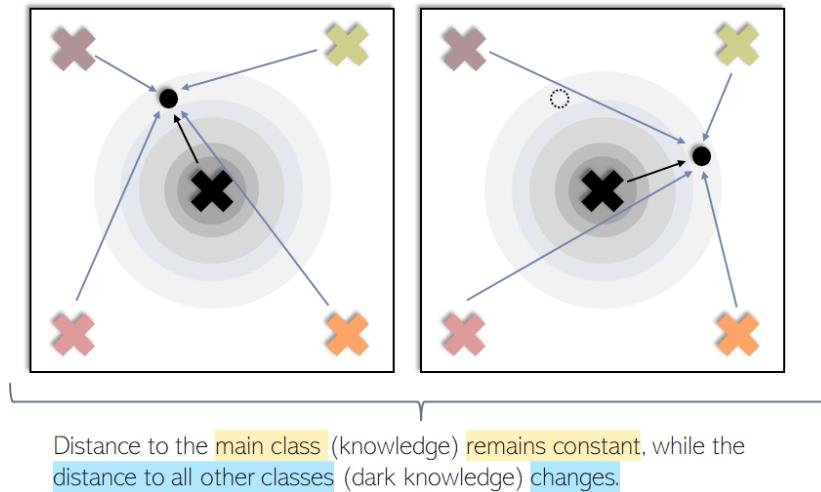


FIGURE 9.20: Illustration of class distances in an embedding space. The graph shows the distance from a point to a class centroid, along with contour lines, representing the main class. These contour lines indicate areas where the distance to the main class remains constant. Moving along these contour lines does not alter the distance to the main class, but the distances to other classes change.

### 9.9.2 Results on Model Output Calibration & Novelty Detection

We compared the calibration discrepancy's outlier detection on synthetic clusters (Table 9.10) and real data (Table 9.11) using class-weighted percentile-based calibrations  $\mathbb{E}_g^\Phi(p_i^{[j]})$ . Real data used specific classes or entire data sets as training sets with outliers regarded as classes not included in the training set or from a completely different data set; while synthetic data had artificial clusters. For real data, experiments were based on large DNNs, while for synthetic data, smaller simple models such as linear perceptrons and decision trees were compared.

TABLE 9.10: Average Out-of-Distribution Area Under the Receiver Operating Characteristic Curve (ROC AUC) scores on *trivial* synthetic clusters for multiple simple classifiers.

algorithm	ROC AUC	algorithm	ROC AUC
ECOD	0.70 ( $\pm 0.01$ )	$\ \epsilon \text{ Perceptron}\ _1^\infty$	0.72 ( $\pm 0.22$ )
Sampling	0.97 ( $\pm 0.03$ )	$\ \epsilon \text{ Perceptron}\ _2^\infty$	0.81 ( $\pm 0.21$ )
QMCD	0.53 ( $\pm 0.07$ )	$\ \epsilon \text{ Perceptron}\ _\infty^\infty$	0.99 ( $\pm 0.01$ )
ABOD	<b>1.00 (<math>\pm 0.00</math>)</b>	$\ \epsilon \text{ SVC}\ _1^\infty$	0.82 ( $\pm 0.22$ )
OCSVM	<b>1.00 (<math>\pm 0.00</math>)</b>	$\ \epsilon \text{ SVC}\ _2^\infty$	0.91 ( $\pm 0.15$ )
MCD	0.89 ( $\pm 0.04$ )	$\ \epsilon \text{ SVC}\ _\infty^\infty$	<b>1.00 (<math>\pm 0.00</math>)</b>
COF	0.44 ( $\pm 0.01$ )	$\ \epsilon \text{ DecisionTreeClassifier}\ _1^\infty$	0.50 ( $\pm 0.08$ )
LOF	<b>1.00 (<math>\pm 0.00</math>)</b>	$\ \epsilon \text{ DecisionTreeClassifier}\ _2^\infty$	0.50 ( $\pm 0.08$ )
HBOS	<b>1.00 (<math>\pm 0.00</math>)</b>	$\ \epsilon \text{ DecisionTreeClassifier}\ _\infty^\infty$	0.53 ( $\pm 0.07$ )
LODA	0.91 ( $\pm 0.03$ )	$\ \epsilon \text{ KNeighborsClassifier}\ _1^\infty$	0.50 ( $\pm 0.03$ )
IForest	<b>1.00 (<math>\pm 0.00</math>)</b>	$\ \epsilon \text{ KNeighborsClassifier}\ _2^\infty$	0.50 ( $\pm 0.03$ )
INNE	0.99 ( $\pm 0.01$ )	$\ \epsilon \text{ KNeighborsClassifier}\ _\infty^\infty$	0.61 ( $\pm 0.03$ )
hdbSCAN	<b>1.00 (<math>\pm 0.00</math>)</b>	$\ \epsilon \text{ GaussianProcessClassifier}\ _1^\infty$	0.76 ( $\pm 0.31$ )
ENSEMBLE	<b>1.00 (<math>\pm 0.00</math>)</b>	$\ \epsilon \text{ GaussianProcessClassifier}\ _2^\infty$	0.80 ( $\pm 0.29$ )
		$\ \epsilon \text{ GaussianProcessClassifier}\ _\infty^\infty$	0.99 ( $\pm 0.01$ )

TABLE 9.11: Average Out-of-Distribution Area Under the Receiver Operating Characteristic Curve (ROC AUC) scores on real-world data. The column "alg\_log" represents the application of the outlier-detection algorithm on the logits, while "alg\_pred" its application on the predictions. The columns "max[alg\_log,  $\epsilon$ ]" and "max[alg\_pred,  $\epsilon$ ]" represent the ROC AUC scores computed based on the maximum of the algorithm and  $\epsilon$ , using either the logits or the predictions, respectively.

Trainset: CIFAR10 - Outlier: MNIST				Trainset: MNIST - Outlier: CIFAR10			
ECOD	alg_log	alg_pred	max[alg_log, $\epsilon$ ]	max[alg_pred, $\epsilon$ ]	alg_log	alg_pred	max[alg_log, $\epsilon$ ]
Sampling	0.31 ( $\pm 0.03$ )	0.31 ( $\pm 0.02$ )	<b>0.78 (<math>\pm 0.06</math>)</b>	<b>0.78 (<math>\pm 0.06</math>)</b>	0.15 ( $\pm 0.02$ )	0.20 ( $\pm 0.02$ )	<b>0.98 (<math>\pm 0.01</math>)</b>
QMCD	0.56 ( $\pm 0.11$ )	0.58 ( $\pm 0.08$ )	<b>0.79 (<math>\pm 0.03</math>)</b>	<b>0.79 (<math>\pm 0.05</math>)</b>	0.84 ( $\pm 0.07$ )	0.85 ( $\pm 0.09$ )	<b>0.98 (<math>\pm 0.02</math>)</b>
ABOD	0.55 ( $\pm 0.04$ )	0.51 ( $\pm 0.09$ )	<b>0.81 (<math>\pm 0.06</math>)</b>	0.80 ( $\pm 0.07$ )	0.48 ( $\pm 0.09$ )	0.39 ( $\pm 0.10$ )	<b>0.99 (<math>\pm 0.01</math>)</b>
OCSVM	0.81 ( $\pm 0.04$ )	0.80 ( $\pm 0.04$ )	<b>0.83 (<math>\pm 0.04</math>)</b>	<b>0.83 (<math>\pm 0.04</math>)</b>	<b>0.99 (<math>\pm 0.01</math>)</b>	<b>0.99 (<math>\pm 0.01</math>)</b>	<b>0.99 (<math>\pm 0.01</math>)</b>
MCD	0.77 ( $\pm 0.06$ )	0.72 ( $\pm 0.07$ )	<b>0.82 (<math>\pm 0.03</math>)</b>	<b>0.82 (<math>\pm 0.03</math>)</b>	<b>0.99 (<math>\pm 0.00</math>)</b>	<b>0.99 (<math>\pm 0.01</math>)</b>	<b>0.99 (<math>\pm 0.00</math>)</b>
COF	0.55 ( $\pm 0.09$ )	0.43 ( $\pm 0.08$ )	<b>0.78 (<math>\pm 0.06</math>)</b>	0.77 ( $\pm 0.06$ )	0.60 ( $\pm 0.01$ )	0.51 ( $\pm 0.01$ )	<b>0.98 (<math>\pm 0.02</math>)</b>
LOF	0.49 ( $\pm 0.01$ )	0.50 ( $\pm 0.01$ )	<b>0.80 (<math>\pm 0.05</math>)</b>	<b>0.80 (<math>\pm 0.05</math>)</b>	0.52 ( $\pm 0.01$ )	0.52 ( $\pm 0.01$ )	<b>0.98 (<math>\pm 0.01</math>)</b>
HBOS	0.66 ( $\pm 0.08$ )	0.65 ( $\pm 0.07$ )	<b>0.82 (<math>\pm 0.03</math>)</b>	0.81 ( $\pm 0.03$ )	0.88 ( $\pm 0.06$ )	0.92 ( $\pm 0.05$ )	<b>0.99 (<math>\pm 0.01</math>)</b>
LODA	0.26 ( $\pm 0.07$ )	0.36 ( $\pm 0.08$ )	0.76 ( $\pm 0.06$ )	<b>0.77 (<math>\pm 0.06</math>)</b>	0.55 ( $\pm 0.19$ )	0.50 ( $\pm 0.19$ )	<b>0.98 (<math>\pm 0.01</math>)</b>
IForest	0.20 ( $\pm 0.07$ )	0.18 ( $\pm 0.05$ )	<b>0.76 (<math>\pm 0.07</math>)</b>	<b>0.76 (<math>\pm 0.07</math>)</b>	0.06 ( $\pm 0.05$ )	0.16 ( $\pm 0.12$ )	<b>0.98 (<math>\pm 0.02</math>)</b>
INNE	0.24 ( $\pm 0.06$ )	0.26 ( $\pm 0.06$ )	<b>0.76 (<math>\pm 0.06</math>)</b>	<b>0.76 (<math>\pm 0.06</math>)</b>	0.45 ( $\pm 0.19$ )	0.35 ( $\pm 0.17$ )	<b>0.98 (<math>\pm 0.01</math>)</b>
hdbscan	0.21 ( $\pm 0.06$ )	0.23 ( $\pm 0.07$ )	<b>0.76 (<math>\pm 0.06</math>)</b>	<b>0.76 (<math>\pm 0.06</math>)</b>	0.18 ( $\pm 0.08$ )	0.23 ( $\pm 0.10$ )	<b>0.98 (<math>\pm 0.01</math>)</b>
Ensemble	0.73 ( $\pm 0.03$ )	0.72 ( $\pm 0.03$ )	<b>0.83 (<math>\pm 0.04</math>)</b>	0.82 ( $\pm 0.04$ )	0.97 ( $\pm 0.02$ )	0.98 ( $\pm 0.02$ )	<b>0.99 (<math>\pm 0.01</math>)</b>
$\ \epsilon\ _1^\infty$			<b>0.84 (<math>\pm 0.04</math>)</b>				<b>0.99 (<math>\pm 0.01</math>)</b>
$\ \epsilon\ _2^\infty$			0.83 ( $\pm 0.05$ )				<b>0.99 (<math>\pm 0.01</math>)</b>
$\ \epsilon\ _\infty^\infty$			0.82 ( $\pm 0.06$ )				<b>0.99 (<math>\pm 0.01</math>)</b>
Trainset: FashionMNIST - Outlier: KMNIST				Trainset: CIFAR10 - Outlier: FashionMNIST			
ECOD	alg_log	alg_pred	max[alg_log, $\epsilon$ ]	max[alg_pred, $\epsilon$ ]	alg_log	alg_pred	max[alg_log, $\epsilon$ ]
Sampling	0.23 ( $\pm 0.02$ )	0.28 ( $\pm 0.02$ )	<b>0.93 (<math>\pm 0.02</math>)</b>	<b>0.93 (<math>\pm 0.02</math>)</b>	0.33 ( $\pm 0.02$ )	0.33 ( $\pm 0.02$ )	<b>0.79 (<math>\pm 0.04</math>)</b>
QMCD	0.69 ( $\pm 0.08$ )	0.67 ( $\pm 0.11$ )	<b>0.94 (<math>\pm 0.02</math>)</b>	0.93 ( $\pm 0.02$ )	0.56 ( $\pm 0.07$ )	0.57 ( $\pm 0.06$ )	<b>0.80 (<math>\pm 0.04</math>)</b>
ABOD	0.56 ( $\pm 0.07$ )	0.41 ( $\pm 0.06$ )	<b>0.96 (<math>\pm 0.01</math>)</b>	0.94 ( $\pm 0.02$ )	0.53 ( $\pm 0.04$ )	0.48 ( $\pm 0.09$ )	<b>0.83 (<math>\pm 0.04</math>)</b>
OCSVM	0.94 ( $\pm 0.01$ )	0.94 ( $\pm 0.01$ )	<b>0.96 (<math>\pm 0.01</math>)</b>	<b>0.96 (<math>\pm 0.01</math>)</b>	0.77 ( $\pm 0.04$ )	0.77 ( $\pm 0.04$ )	<b>0.83 (<math>\pm 0.03</math>)</b>
MCD	0.92 ( $\pm 0.02$ )	0.92 ( $\pm 0.02$ )	<b>0.95 (<math>\pm 0.01</math>)</b>	<b>0.95 (<math>\pm 0.01</math>)</b>	0.74 ( $\pm 0.04$ )	0.69 ( $\pm 0.04$ )	<b>0.81 (<math>\pm 0.03</math>)</b>
COF	0.55 ( $\pm 0.02$ )	0.49 ( $\pm 0.03$ )	<b>0.93 (<math>\pm 0.02</math>)</b>	<b>0.93 (<math>\pm 0.02</math>)</b>	0.46 ( $\pm 0.06$ )	0.44 ( $\pm 0.11$ )	<b>0.78 (<math>\pm 0.04</math>)</b>
LOF	0.49 ( $\pm 0.01$ )	0.49 ( $\pm 0.01$ )	<b>0.94 (<math>\pm 0.02</math>)</b>	<b>0.94 (<math>\pm 0.02</math>)</b>	0.53 ( $\pm 0.01$ )	0.52 ( $\pm 0.01$ )	<b>0.81 (<math>\pm 0.03</math>)</b>
HBOS	0.81 ( $\pm 0.02$ )	0.85 ( $\pm 0.03$ )	<b>0.95 (<math>\pm 0.01</math>)</b>	<b>0.95 (<math>\pm 0.02</math>)</b>	0.64 ( $\pm 0.04$ )	0.65 ( $\pm 0.04$ )	<b>0.81 (<math>\pm 0.03</math>)</b>
LODA	0.40 ( $\pm 0.09$ )	0.50 ( $\pm 0.15$ )	<b>0.93 (<math>\pm 0.02</math>)</b>	<b>0.93 (<math>\pm 0.02</math>)</b>	0.25 ( $\pm 0.04$ )	0.35 ( $\pm 0.07$ )	<b>0.78 (<math>\pm 0.04</math>)</b>
IForest	0.05 ( $\pm 0.02$ )	0.15 ( $\pm 0.09$ )	<b>0.92 (<math>\pm 0.02</math>)</b>	<b>0.92 (<math>\pm 0.02</math>)</b>	0.18 ( $\pm 0.03$ )	0.20 ( $\pm 0.06$ )	<b>0.76 (<math>\pm 0.05</math>)</b>
INNE	0.19 ( $\pm 0.06$ )	0.36 ( $\pm 0.10$ )	<b>0.93 (<math>\pm 0.02</math>)</b>	0.92 ( $\pm 0.02$ )	0.22 ( $\pm 0.03$ )	0.23 ( $\pm 0.07$ )	<b>0.76 (<math>\pm 0.04</math>)</b>
hdbscan	0.12 ( $\pm 0.03$ )	0.35 ( $\pm 0.07$ )	<b>0.93 (<math>\pm 0.02</math>)</b>	<b>0.93 (<math>\pm 0.02</math>)</b>	0.21 ( $\pm 0.04$ )	0.24 ( $\pm 0.04$ )	<b>0.76 (<math>\pm 0.05</math>)</b>
Ensemble	0.85 ( $\pm 0.05$ )	0.86 ( $\pm 0.04$ )	<b>0.95 (<math>\pm 0.01</math>)</b>	<b>0.95 (<math>\pm 0.01</math>)</b>	0.70 ( $\pm 0.04$ )	0.69 ( $\pm 0.04$ )	<b>0.82 (<math>\pm 0.03</math>)</b>
$\ \epsilon\ _1^\infty$			<b>0.96 (<math>\pm 0.01</math>)</b>				<b>0.84 (<math>\pm 0.03</math>)</b>
$\ \epsilon\ _2^\infty$			0.95 ( $\pm 0.01$ )				0.83 ( $\pm 0.03$ )
$\ \epsilon\ _\infty^\infty$			0.95 ( $\pm 0.01$ )				0.82 ( $\pm 0.04$ )
Trainset: KMNIST Classes 1-5 - Outlier: KMNIST Classes 6-10				Trainset: CIFAR Classes 1-5 - Outlier: CIFAR Classes 6-10			
ECOD	alg_log	alg_pred	max[alg_log, $\epsilon$ ]	max[alg_pred, $\epsilon$ ]	alg_log	alg_pred	max[alg_log, $\epsilon$ ]
Sampling	0.28 ( $\pm 0.01$ )	0.31 ( $\pm 0.02$ )	0.74 ( $\pm 0.04$ )	<b>0.75 (<math>\pm 0.04</math>)</b>	0.43 ( $\pm 0.01$ )	0.44 ( $\pm 0.01$ )	0.61 ( $\pm 0.02$ )
QMCD	0.73 ( $\pm 0.11$ )	0.82 ( $\pm 0.06$ )	0.82 ( $\pm 0.05$ )	<b>0.85 (<math>\pm 0.04</math>)</b>	0.55 ( $\pm 0.04$ )	0.54 ( $\pm 0.04$ )	<b>0.64 (<math>\pm 0.03</math>)</b>
ABOD	0.51 ( $\pm 0.05$ )	0.38 ( $\pm 0.14$ )	<b>0.80 (<math>\pm 0.03</math>)</b>	0.78 ( $\pm 0.04$ )	0.55 ( $\pm 0.08$ )	0.47 ( $\pm 0.09$ )	<b>0.68 (<math>\pm 0.04</math>)</b>
OCSVM	0.91 ( $\pm 0.01$ )	<b>0.91 (<math>\pm 0.01</math>)</b>	0.84 ( $\pm 0.02$ )	0.84 ( $\pm 0.02$ )	0.59 ( $\pm 0.02$ )	0.59 ( $\pm 0.01$ )	<b>0.66 (<math>\pm 0.02</math>)</b>
MCD	0.89 ( $\pm 0.02$ )	<b>0.90 (<math>\pm 0.02</math>)</b>	0.89 ( $\pm 0.02$ )	<b>0.90 (<math>\pm 0.02</math>)</b>	0.57 ( $\pm 0.01$ )	0.57 ( $\pm 0.01$ )	<b>0.64 (<math>\pm 0.01</math>)</b>
COF	0.59 ( $\pm 0.02$ )	0.54 ( $\pm 0.03$ )	<b>0.77 (<math>\pm 0.03</math>)</b>	0.75 ( $\pm 0.03$ )	0.51 ( $\pm 0.03$ )	0.47 ( $\pm 0.04$ )	<b>0.62 (<math>\pm 0.03</math>)</b>
LOF	0.51 ( $\pm 0.01$ )	0.50 ( $\pm 0.01$ )	<b>0.80 (<math>\pm 0.03</math>)</b>	<b>0.80 (<math>\pm 0.03</math>)</b>	0.49 ( $\pm 0.01$ )	0.49 ( $\pm 0.01$ )	<b>0.63 (<math>\pm 0.02</math>)</b>
HBOS	0.73 ( $\pm 0.05$ )	0.75 ( $\pm 0.05$ )	<b>0.82 (<math>\pm 0.03</math>)</b>	<b>0.82 (<math>\pm 0.03</math>)</b>	0.53 ( $\pm 0.02$ )	0.53 ( $\pm 0.02$ )	<b>0.63 (<math>\pm 0.02</math>)</b>
LODA	0.72 ( $\pm 0.09$ )	0.74 ( $\pm 0.07$ )	0.79 ( $\pm 0.04$ )	<b>0.82 (<math>\pm 0.04</math>)</b>	0.45 ( $\pm 0.02$ )	0.50 ( $\pm 0.03$ )	0.59 ( $\pm 0.03$ )
IForest	0.17 ( $\pm 0.07$ )	0.59 ( $\pm 0.22$ )	0.74 ( $\pm 0.04$ )	<b>0.81 (<math>\pm 0.07</math>)</b>	0.33 ( $\pm 0.02$ )	0.34 ( $\pm 0.03$ )	<b>0.57 (<math>\pm 0.03</math>)</b>
INNE	0.68 ( $\pm 0.06$ )	0.70 ( $\pm 0.08$ )	0.77 ( $\pm 0.04$ )	<b>0.78 (<math>\pm 0.04</math>)</b>	0.43 ( $\pm 0.03$ )	0.42 ( $\pm 0.03$ )	<b>0.59 (<math>\pm 0.02</math>)</b>
hdbscan	0.78 ( $\pm 0.06$ )	0.81 ( $\pm 0.07$ )	0.83 ( $\pm 0.04$ )	<b>0.85 (<math>\pm 0.05</math>)</b>	0.44 ( $\pm 0.03$ )	0.45 ( $\pm 0.03$ )	<b>0.59 (<math>\pm 0.03</math>)</b>
Ensemble	0.87 ( $\pm 0.03$ )	0.85 ( $\pm 0.03$ )	<b>0.90 (<math>\pm 0.02</math>)</b>	0.89 ( $\pm 0.02$ )	0.58 ( $\pm 0.01$ )	0.57 ( $\pm 0.02$ )	<b>0.65 (<math>\pm 0.02</math>)</b>
$\ \epsilon\ _1^\infty$			0.85 ( $\pm 0.02$ )	<b>0.87 (<math>\pm 0.02</math>)</b>	0.56 ( $\pm 0.03$ )	0.56 ( $\pm 0.03$ )	<b>0.61 (<math>\pm 0.03</math>)</b>
$\ \epsilon\ _2^\infty$			0.83 ( $\pm 0.03$ )				<b>0.67 (<math>\pm 0.02</math>)</b>
$\ \epsilon\ _\infty^\infty$			0.85 ( $\pm 0.03$ )				0.67 ( $\pm 0.02$ )
			<b>0.91 (<math>\pm 0.02</math>)</b>				0.65 ( $\pm 0.02$ )

Table 9.10 demonstrates several interesting characteristics of the epsilon score and its dependence on the respective classifier. The epsilon score is dependent on the classifier because each classifier has a different accuracy and output distribution, meaning that they assign confidence scores differently. This is important to consider when performing outlier detection here, as the effectiveness of the calibration misalignment method may vary depending on the quality of the classifier used. For instance, it is not sensible to perform outlier detection if the model is not even able to perform well on the training or test set.

On simple synthetic data with well-defined clusters and random noise outside the clusters, most of the conventional methods tested performed (extremely) well with about perfect outlier detection scores. However, the  $\|\epsilon\|$  score struggled in this setting. One possible explanation for this is that noise was defined as random points in the embedding space outside the clusters. In DNNs, however, noise may instead be mapped/embedded using a specific logic; and its intermediate representations may not be simply random. I.e., even though the input noise to the model may be random, after passing through multiple deep layers, the output is not as random anymore. For example, if data is generated from a multivariate Gaussian distribution and passed through a DNN to retrieve the embeddings, the value distributions of the embedding features will not really follow a Gaussian distribution any more (NNs are heavily non-linear); i.e. the DNN model has the ability to transform the input data in a manner that can produce embeddings with a distribution that differs from the original input data. The exact (distributional) mapping of inputs to embeddings is non-trivial and highly dependent on the model, training objective, input data, etc. The model itself defines this mapping.

However, the success of the  $\|\epsilon\|$  score on complex real-world input data in a DNN setting, where identifying outliers in the deep embedding space is challenging, can possibly be attributed to the "non-random" and "non-trivial" mapping of DNNs. This is because the embedding now relies on the latent manifold of the model, and outliers are represented by points that lie outside the corresponding (local) manifolds. Notably, the  $\|\epsilon\|$  score outperformed most traditional methods in this context. In contrast, random outlier points in the input data are, on the other hand, determined e.g. solely by large Euclidean distances or drawn from the tail distributions of the feature values.

Moreover, we noticed that the infinity norm exhibited better performance for DNNs on the synthetic data compared to its overall performance on the real-world data sets (Table 9.11). The infinity norm represents the maximum absolute difference between the features of two vectors. In this particular case, the maximum difference proved to be more informative for outlier detection, possibly due to the outliers being generated by Euclidean distant points initially. This implies that different norms can be more effective in specific scenarios.

Therefore, in summary, we experienced the best results for  $\|\epsilon\|$  on real-world data sets with DNNs, and less on synthetic data sets using simple classifiers. A visual representation of the 2D projection displaying the embeddings of both inliers and outliers within each individual data set is shown in Figure 9.21.

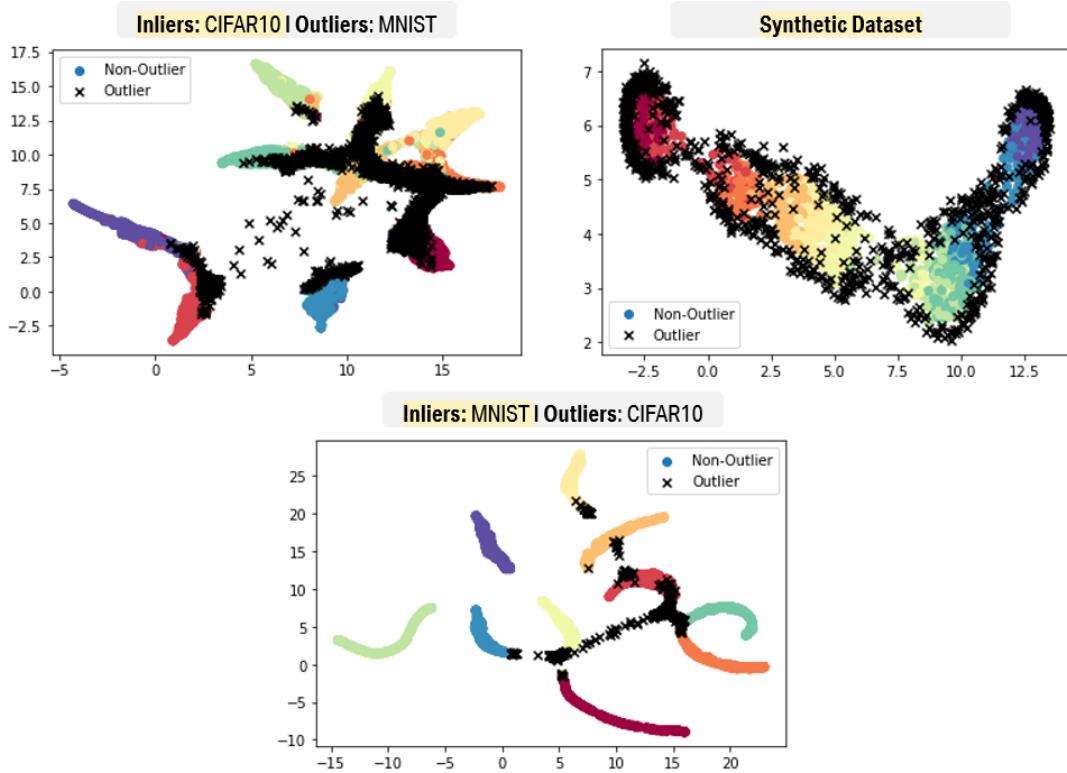


FIGURE 9.21: An embedding projection showing inliers and outliers.

### Take-Away

We evaluated our proposed Out-of-Distribution (OOD) detection method for identifying outliers in DNNs by exploiting differences in output calibration (i.e. misalignment). We found that our method performed better than many traditional algorithms on real-world datasets for DNN-based classification tasks. However, we observed that the method struggled with simpler models like decision trees and had reduced effectiveness on synthetic data with trivial outliers. We believe this difference is because random noise in simpler models does not follow the same complex (manifold) patterns observed in DNNs, where outliers can be identified by their deviation from the (learned) latent manifold space, whereas in simpler models, the input-to-output mapping is inherently different. This has implications for the calibration alignment; on which our OOD method is based. Moreover, the effectiveness of our outlier detection method directly depends on the quality of the model itself for the classification task (effective OOD detection assumes good downstream task performance). This, in turn, is influenced by the quality of the (latent) embedding representation (upon which the traditional OOD detection algorithms directly depend). Overall, proper model quality is essential.

### Keywords

Calibration Misalignment • Out-of-Distribution • Deep Neural Networks

## 9.10 Clustering Strategy: Prediction-based Sub-Clustering

The iterative sub-clustering approach is demonstrated using two data sets: Sklearn's diabetes and California housing data. To start, the data is standardized to ensure consistency. The standardized data is then visualized using UMAP. Next, HDBSCAN is utilized to identify clusters and outliers. Finally, the clusters that have been identified are then again re-clustered. This yields new sub-clusters as well as new outliers (which we categorize as non-core points; since these outliers are only considered outliers within the second iteration of clustering and not within the entire data set). This is illustrated in the 2D plots shown in Figures 9.22 and 9.23 below:

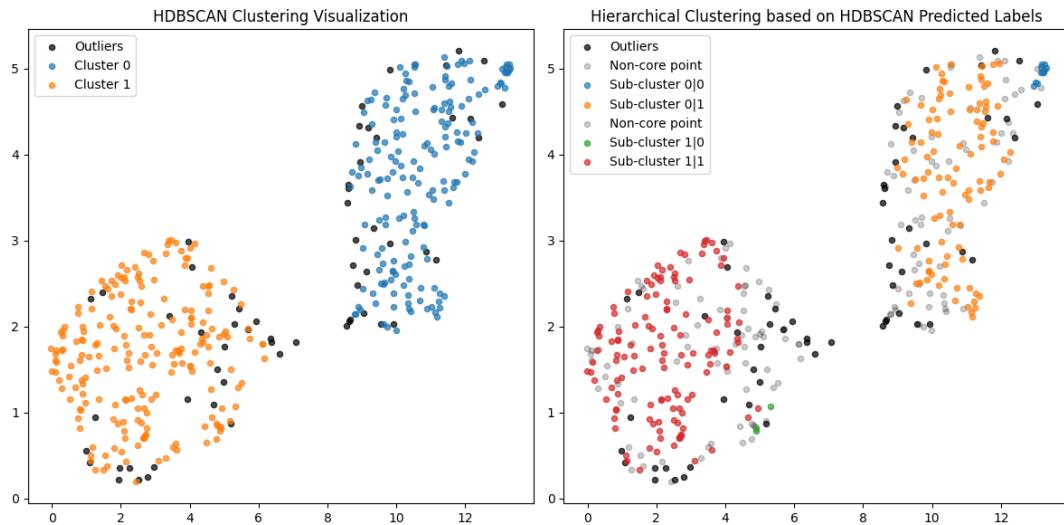


FIGURE 9.22: Analyzing sub-cluster data hierarchies in Sklearn's diabetes data set; identifying core points, clusters, and outliers.

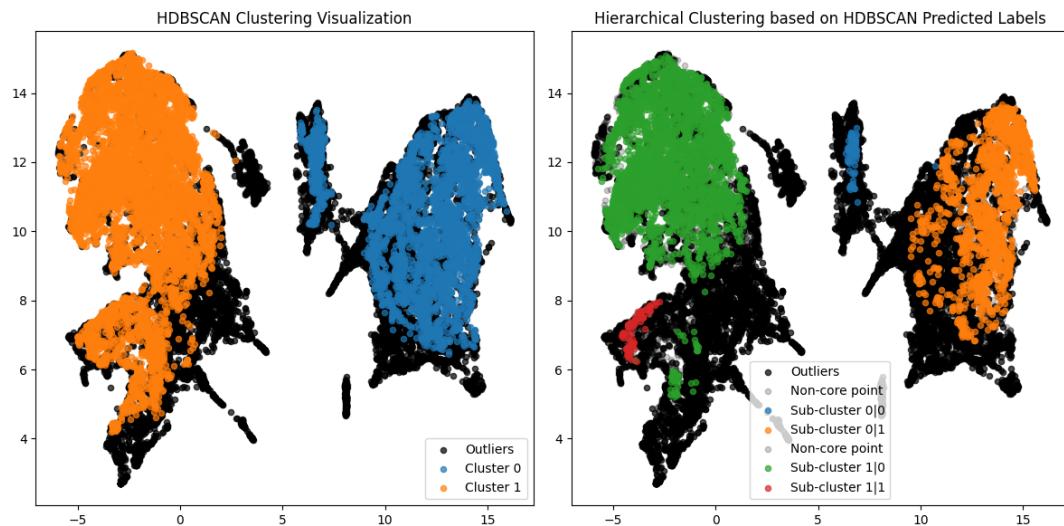


FIGURE 9.23: Analyzing sub-cluster data hierarchies in Sklearn's California housing data set; identifying core points, clusters, and outliers.

An initial clustering reveals primary clusters and outlier points. Outlier points are simply those not associated with any identified cluster. For successive iterations,

clustering occurs within clusters (or prediction labels). This further helps in identifying densely grouped points that form sub-clusters. Notably, not all points of a super-cluster are reassigned to a sub-cluster. Those that are reassigned can be seen as core points, contributing to more defined sub-patterns (i.e. sub-clusters). Conversely, points that remain unassigned to any sub-cluster can be seen as adhering to a more general (coarser) pattern.

In other words, within identified clusters, further sub-clustering is performed using the cluster labels of the first iteration. This helps to uncover additional sub-clusters and non-core points, which can be seen as outliers only within the respective sub-clustering. Consequently, iterative sub-clustering extracts deeper and more homogeneous patterns within the data. Iterative sub-clustering can be done by e.g. sub-clustering the identified clusters using an unsupervised approach, or alternatively, by clustering and sub-clustering based on the class-label prediction of a classifier as a guided supervised approach. If employing prediction-based clustering (i.e. clustering only within a mask of equally predicted labels from a classifier), we can ensure that all clusters and sub-clusters always share the same (classification) labels. This, in particular, prevents mixing different labels within the same cluster, and so, guaranteeing label coherence (i.e. finding patterns only for data points sharing a common label).

Overall, repeated sub-clustering easily facilitates subgroup identification within clusters and improves the detection of core points (i.e. patterns) that exhibit a sufficient number of neighboring points within a certain distance defined by the clustering algorithm (during sub-clustering) and, thus, retain their status as clustering points. Core points are of special interest since they represent dense and homogeneous groups within the data.

### Take-Away

Prediction-based clustering ensures that labels within a cluster are consistent. Through prediction-based clustering and recursive sub-clustering, we can uncover (deeper) nested finer patterns within larger and more general patterns. This can be used to identify dense and homogeneous groups within the data, which can be seen as "core points".

### Keywords

Sub-Clustering • Prediction-based Grouping • Homogeneous Patterns



## Chapter 10

# Discussion

### 10.1 Neural Expressive Power & Bi-Nonlinear Complex-valued Based Neural Networks

**Notice:** Our work and findings on complex-valued NNs (presented in this thesis) have been published in Guimerà Cuevas, Phan, and Schmid, 2023. It was inspired and builds upon our previous research (Guimerà Cuevas and Phan, 2021), but differs significantly as it has been substantially improved and modified (both the complex architecture and its mathematical framework); thus, constitutes novel contributions. In particular, this novel contribution has been accepted for publication, after peer review. The Version of Record is available online at: [https://link.springer.com/chapter/10.1007/978-3-031-33374-3\\_28](https://link.springer.com/chapter/10.1007/978-3-031-33374-3_28). Use of this Version is subject to the publisher's Manuscript terms of use. Therefore, please refer to Guimerà Cuevas, Phan, and Schmid, 2023 for specific details.

In this thesis, we introduced a novel neural architecture with unique expressive characteristics compared to traditional real-valued models. The "expressive power" of a neural network, crucial for learning complex relationships and generalization, is often enhanced by increasing model size and scaling up architectures. However, it is equally important to explore new ways to define the underlying neural architecture, as seen in innovations like liquid time-constant networks (Hasani et al., 2021), structured state spaces (S4) (Gu, Goel, and Ré, 2021), and the recent *Mamba* architecture (Gu and Dao, 2023).

Our approach focuses on refining the forward and backward passes of a model rather than merely scaling up. We proposed a bi-non-linear neural network based on complex numbers, where neural connections are confined within the unit circle, preventing them from being nullified or excessively amplified. This property allows for the preservation of identity in embeddings and demonstrates effectiveness across tasks, sometimes outperforming real-valued models; see Chapters 9.5 and 9.1.

#### Take-Away

Improving neural network expressiveness is crucial, but evaluating architectures fairly is a challenge. Utilizing bi-nonlinear networks with complex numbers confined in unit circles showed interesting results; and constitutes (explores) a different approach to neural manifold learning.

#### Keywords

Neural Expressive Power • Complex Numbers • Model Architecture

## 10.2 Hierarchical Heterogeneous Graph Neural Networks for Deep Learning on Recursive One-to-Many Databases

The HDB-model was successful in embedding DB (Rebrickable®, 2022) instances in such a way that items with similar characteristics were located close to each other (and dissimilar ones far apart). This resulted in clusters of objects with similar pieces (i.e. sets) and comparable part counts. As a result, many of the instances within a cluster were also visually similar, even though the embeddings were calculated explicitly without considering visual features; nearby instances on the manifold embedding had many common and similar Rebrickable® pieces, e.g. from similar theme sets. By mapping similar DB instances together, we can find patterns that are not apparent at first. In particular, our model has the potential to explore relationships and patterns between entities in different HDB settings, including fully unsupervised, semi-supervised, self-supervised, and supervised scenarios. Overall, patterns are identified based on both, the topological tree structure of the DB entity, as well as the node attributes themselves; using the entire tree structure and all features.

In general, we found that residual connections were advantageous in downstream tasks and that the performance of a semi-supervised multi-objective classification with an identity preservation task was slightly improved by using the proposed kernel embedding. However, in a simple classification-only task, the use of a kernel embedding resulted in worse performance, especially for the synthetic data. This is due to identity preservation having a potential counteracting effect on supervised class classification (i.e. efficiency). Since identity preservation may not be necessary in this context, it may add unnecessary complexities that affect the model’s performance and learning convergence.

By including an identity preservation constraint, we aim to create embeddings that not only have high quality but also maintain information about the entity. This means that the embeddings will encode information about the identity of the entities in the data and so be useful and effective for multiple combined (different) downstream tasks, such as classification with (sub-)clustering. For example, the goal might be to obtain an embedding that not only has a low reconstruction error after being processed by a decoder but also maps similar objects close in the embedding space, distinguishes between different objects in a contrastive manner, classifies these accurately, and is further capable of identifying semantic anomalies (through embedding clustering).

We analyzed and compared our proposed method against a naive Transformer-based encoding scheme that encodes HDB paths in a textual format from root to leaf nodes, without positional encoding, random token ordering, and dropped token duplicates. The brute-force approach of the naive Transformer-based encoding scheme has many important limitations. Transformers are most suitable for token-based inputs such as categorical data and are not intended to work directly with numeric features. In Transformers, numeric features are mapped to textual tokens, which increases the feature dimension during conversion (for instance, the number ‘123456’ might be tokenized into ‘12’, ‘34’, and ‘56’, each with a full token embedding dimension of e.g. 512). Our proposed method, on the other hand, processes both numeric features and textual data directly/separately, without such need for a numeric re-mapping (unless they occur in the text; but what we here actually refer to are the numeric features of a given HDB). This avoids the (sub-optimal) transformation steps otherwise required to convert the numeric data to textual tokens. Moreover, Transformers have a pre-defined maximum input sequence length, which

can be problematic in cases where the brute-force textual (Transformer-based) token encoding of a given HDB is too large. This is quickly often the case, especially for larger HDBs that contain multiple tables, features, and relations; the textual input sequence must then be truncated (which is bad, as it results in information loss). To simulate this, additionally, we also randomized the token ordering during sampling and then applied a maximum sequence limit on it of around the 0.767 percentile of the overall sequence-length distribution (of all HDBs) for Bert; and of around 0.287 for \*Bert. We included this comparison in our evaluation against the proposed HDB-models (in addition to the *non*-constrained Transformer-based brute-force baseline).

During the evaluation, we observed that the HDB-model performed generally much better in training scores and convergence speed compared to the brute-force Bert-based Transformer model. However, the HDB-model tended to overfit the data more quickly, indicating the need for stronger regularization, and which could sometimes lower the performance on the test set (scores compared to Bert). One (small) difference between the two models (HDB vs. Bert) was that for the HDB-model we only used standard attention instead of a multi-head attention mechanism (as in Bert); although one could easily incorporate it if desired. However, this was not of relevant or crucial consequence, since both architectures were already inherently completely different (recursive HDB vs. Transformer-based Bert). We also used standard real-valued NNs instead of our proposed complex-valued networks (specifically, utilizing a real-valued conventional matrix feed-forward pass with residual connections) for better and more direct architecture comparison. If we view the feed-forward architecture as a self-contained module in the overall HDB-model structure (black-box), it is evident that the performance of the HDB-model can always be enhanced through targeted hyper- and architecture optimization of these black-box modules. We can think of the recursive HDB model structure as a hierarchical composition of such modules, resembling a series of interconnected self-contained black-boxes.

We noticed that the Bert models were more sensitive to the learning rate choice, leading often even to total training failures with high learning rates; while the HDB-models were very robust, especially HDB-v2. Limiting the maximal token input sequence size for the Bert models, as expected, greatly negatively impacted the classification effectiveness and convergence, resulting in a significant performance decrease. This limitation becomes especially concerning for real-world applications, since large databases will not fit within the specified maximum sequence length, requiring truncation of tokens and subsequent decrease in performance (again, due to information loss). In contrast, the HDB-models handle the entire DB tree with all features in a hierarchical manner, avoiding this issue; i.e. they do *not* lose information due to truncation.

In summary, we introduced a method for applying ML and DL to complex large HDBs that have recursive one-to-many relations. Our method, particularly, allows us to compute strong embedded representations for all entities in the HDB and enables, so, many downstream tasks like classification, clustering, anomaly detection, etc. It has clear advantages compared to a brute-force Transformer-based textual encoding approach. Our approach consists of representing HDB relationships as edges and the tables as nodes (as a hierarchical tree-like structure). The encoding and model scheme is not only limited to HDBs, but can be applied to any data structure that can be transformed into a hierarchical tree with loop-free connections. The only requirement is that node features under a common parent must be equivalent (otherwise a virtual node must be introduced); but they can differ completely across other nodes at different hierarchy levels. In particular, our solution directly supports node

entities (DB tables) that have different features; which is not typically supported in standard GNNs. Compared to naive Transformer-based encoding approaches that encode the HDB as text, our proposed HDB-model architecture offers two key differences (advantages): it can directly process numeric features, and avoids the truncation of long encoded token sequences. Experiments with real-world and synthetic data demonstrated the effectiveness of our method. We found that adding residual connections improved performance in purely supervised classification tasks, while the use of a Kernel embedding improved performance in semi-supervised tasks that require identity and information preservation.

### Take-Away

We introduced a learning approach, the HDB-model, specifically designed for hierarchical databases. Our method represents relationships within HDBs as nodes and edges, enabling the direct application of ML techniques to these complex data structures. Our HDB-model effectively performs downstream tasks such as clustering, anomaly detection, classification, etc. The HDB-model offers several advantages: robustness in different learning rates, accommodates input sequences of any size, recursively processes the entire database, etc. However, it also tends to experience overfitting more easily and thus requires adequate regularization or training techniques.

### Keywords

Hierarchical Databases • Machine- & Deep Learning • Model Architecture

## 10.3 Leveraging Knowledge Distillation and Domain Adaptation in Training for Transfer Learning

Knowledge transfer typically involves transferring knowledge from a larger model to a smaller one. It is a very common and standard practice; known as knowledge distillation. In this thesis, we focused not only on transferring knowledge from one model to another but also on transferring knowledge from a less powerful (interpretable model) to a stronger (non-interpretable) intermediate model, to then extract again a final smaller interpretable model (from the intermediate one). This was motivated by the need to shift to a new input domain, such as a completely different data set. Consequently, we had to transfer knowledge not only across *models*, but also, across *input domains*.

For that, we proposed an approach to such knowledge transfer across different domains; i.e. for models trained on different data sets. The goal was to transfer knowledge from a model trained on a (labeled) straightforward data set with intuitive features to an intermediate model trained on a more complex yet highly precise database. Ultimately, a final new interpretable model from this intermediate model was extracted, using only the intricate and complex data from the new database; i.e. *solely* drawing its rules from the complex data set. However, extracting a simpler model from an intermediate model leads to information loss (e.g. of specific details). The quality of the final model relies heavily on that of the intermediate model, and challenges in extracting rules must be carefully addressed to ensure success. For instance, even if the intermediate model is "perfect", a poorly designed rule extraction algorithm will result in poor final outputs (i.e. rules). Likewise, if the knowledge

distillation process from the first model is weak, it will affect the quality of the intermediate model and, consequently, the final model. Also, potential biases present in the original data set or intermediate model may persist or even be magnified in the final simplified model, e.g. also leading to biased or inaccurate outcomes.

Our approach, in particular, can also be used to (just) change the domain of the training data (no intermediate or final interpretable model); e.g. we have a data set with predicted labels for a certain subset of features, but now we want to use this knowledge to predict labels for a different subset of features. In real-world scenarios, it is crucial to adapt models to technological advancements and new data. This is especially important when dealing with evolving data sets and rapidly improving or changing technology; e.g. we may repeatedly need to update existing models to make use of (new) information obtained by using new/different (e.g. more precise, robust, etc.) hardware, and sensors; especially, if access to previous data sources (e.g. hardware) is not possible anymore and the data input domains differ.

For instance, suppose we initially collected data using specific tools or sensors, which allowed us to predict and categorize certain features. However, over time, more advanced sensors have become available that enable us to gather a new set of data measurements. These new sensors can measure properties that were previously impossible or now with even greater accuracy. To smoothly transition between the different data domains and make use of our previous knowledge, we use cross-domain adaptation: incorporating insights and knowledge from our previous model that was based on old data. Then, when training a new model on the new dataset, we do *not* need to re-learn or re-extract the labels and dark knowledge (Hinton, Vinyals, and Dean, 2015) anew. This is very valuable since it saves time and resources (and potential errors).

#### Take-Away

Transferring knowledge across different domains and models is often required. By performing knowledge distillation, models can be adjusted to incorporate new or updated data; while still pertaining to (i.e. maintaining) old knowledge. The goal is to leverage/utilize valuable previous (dark) knowledge; without having to re-learn everything anew.

#### Keywords

Cross-Domain Knowledge Transfer & Distillation • Domain Adaptation

## 10.4 Robust Non-linear Normalization of Heterogeneous Feature Distributions with Adaptive Tanh-Estimators

**Please note:** This contribution has been accepted for publication, after peer review in Guimerà Cuevas and Schmid, 2024b. The Version of Record is available online.

For ML models to be effective, feature pre-processing (FP) is often fundamental. The actual FP technique used is crucial, and an inadequate FP method can have a severe negative impact on the training convergence and final performance; especially for ML models. Although tanh-normalization gives robust outlier resilience, it does so by using a fixed spread value for all features during normalization. This is *not* ideal but can be improved by instead automatically selecting a spread factor so that the FP output follows a desirable (ideal) output distribution. Good spread

values improve training convergence, or likewise, poor spread values decrease performance substantially. Our proposed adaptive Wasserstein-tanh normalization not only finds the ideal spread values but also eliminates the need for manual parameter tuning by automatically determining and selecting these optimal spread values. This saves time and removes an additional manual hyper-parameter tuning step.

Overall, pre-processing of feature values is important to ensure optimal model performance. For numeric feature vector representations, it is crucial to ensure that the ranges of feature values are comparable so that no single feature dominates the others; which e.g. could cause bias. FS is often used here to remap the individual features into comparable (unit) ranges. It can, however, be sensitive to outliers in the data and even outliers in the individual feature dimensions. *Tanh*-estimators offer robust feature normalization here, but as explained, utilize a fixed scaling factor for all features; which raises the question of whether this is suitable given the vast variation of possible feature distributions.

Via a series of tests, we empirically confirmed that poorly spread density masses of normalized feature value distributions were to blame for slow training convergence. When a fixed global spread value is used to normalize all features, this is expected to happen, particularly e.g. if feature distributions vary greatly. This thesis proposed an adaptive form of the tanh-normalization that calculates the ideal spread based on minimizing the Wasserstein distance of feature distributions against a target distribution to control the probability density of the normalization. The method particularly enhances the literature's current method, boosting training convergence speed by avoiding unnecessary model weight adaption due to poor FS. We repeatedly highlighted the importance of determining ideal *feature-wise* spread values and provided theoretical motivation. Our solution determines the ideal spreads automatically and effectively, avoiding time-consuming manual adjustments. Just as the traditional tanh-normalization, it is robust to outliers, efficient, and very simple to incorporate, but further ensures a normalizing transformation that gives an effective probability mass distribution.

Therefore, the importance and role of FP must not be underestimated. Inadequate FP methods can strongly negatively affect model performance; as demonstrated and highlighted. Improper normalization or scaling of features can cause many problems, such as model bias, slow training convergence, lower effective performance, etc.

### Take-Away

Feature pre-processing (FP) is crucial for optimal model performance. Tanh-normalization is a popular FP technique, which is robust to outliers but can be greatly improved by automatically selecting the most ideal spread that best results in an optimal/desirable output distribution; instead of using a fixed global spread for all feature distributions. This improves training convergence substantially and eliminates the need for manual parameter tuning.

### Keywords

Pre-processing • Tanh-Normalization • Outliers • Feature Distribution

## 10.5 Supervised $\alpha$ Max- $B^3$ Clustering Evaluation Metric

**Notice:** Our work and findings on  $\alpha$ Max-B-CUBED (presented in this thesis) have been publicly released on OpenReview-Venue (Guimerà Cuevas and Schmid, n.d.).

Coarse ground-truth labels make it harder to accurately evaluate clustering results. A poor or wrong evaluation measure/metric can certainly confuse the analyst and lead to incorrect interpretations and conclusions (e.g. the choice of the right model, hyper-parameters, etc). Being aware of the issue enables us to counteract or avert this situation. The fundamental challenge is how to adequately evaluate clustering results using a supervised metric if the labels are coarse and not totally representative of ground truths. We approached this problem by leveraging the observation that we can determine or choose a degree of uncertainty to either encourage or discourage sub-group identification. Even though the labels are inexact, using a supervised measure is still advantageous in this scenario. Recall, that an unsupervised loss function is often based on the assumption that data points within a cluster are similar to each other but dissimilar to those in other clusters. This may not be appropriate here since sub-groups can be extremely similar (yet still different); and importantly, it does not guarantee that clusters are homogeneous: we do not want to mix instances with different labels in the same cluster. If the labels were exactly the ground truth, or if we were not concerned with discovering sub-groups, setting the uncertainty  $\alpha$  to zero (in our  $\alpha$ Max- $B^3$ ) will yield the same results as  $B^3$ .

The  $\alpha$  parameter in  $\alpha$ Max- $B^3$  is cluster-specific because it is dependent on the precision and recall of a particular cluster.  $\alpha$  and  $\eta_\alpha$  are related, although they have distinct roles: the final cluster uncertainty score is based on precision and recall and reflected by  $\eta_\alpha$ ; on the other hand,  $\alpha$  reflects the uncertainty between the original clusters and the merged super-clusters, and  $\eta_\alpha$  then uses  $\alpha$  for its score evaluation. The values for  $\alpha$  can also be set manually if desired, e.g. if the degree of uncertainty is known a priori; otherwise, however, manual adjustment implies hyper-parameter tweaking, which generally is time-consuming and non-trivial. Thus, we recommend using our proposed default automatic uncertainty determination; our experiments have shown that this uncertainty determination gives a good estimation.

That is, the automatic determination of  $\alpha$ , selected over the function's maxima Plateau (see Figure 7.33), approximates the real (unknown) uncertainty in the data well, and about perfectly over a consistent set of uncertainty across all clusters. However, the more noisy and uneven the cluster uncertainty is across the clusters (i.e. for different levels of uncertainties across clusters), the less accurate this approximation becomes. Still, it provides a good automatic way of selecting an appropriate uncertainty value. In other words, we can extract the level of uncertainty by measuring the plateau interval and extrapolating the corresponding  $\alpha$  values to obtain an approximation or estimate of the data set uncertainty.

Overall, we have highlighted an inherent issue concerning the "completeness" criterion used in the conventional  $B^3$  cluster quality metric when coarse labels are involved, which can be problematic, unfair, and misleading to evaluative (human) judgment. To counteract this issue, a novel solution,  $\alpha$ Max- $B^3$ , has been proposed, which addresses this potential problem of unfair and inexact evaluation inherent in the standard  $B^3$  metric. A comprehensive analysis of its properties and a formal description have been provided, and a new clustering metric formula has been devised that allows for a more fair comparison. The formula is capable of adapting to sub-group uncertainty in ground-truth labels and can be generalized to accommodate imbalanced data sets. Our method first merges clusters based on their most

frequent label into larger groups called super-sets. These super-sets are then evaluated in conjunction with the original clusters using a modified  $B^3$  metric, which applies a weighting factor to control the contribution of the super-sets against the original clusters. Unlike the standard  $B^3$  metric, which typically favors coarse clusters above potentially valid sub-clusters,  $\alpha$ Max- $B^3$  produces more robust and fair results; accounting for super-class (label) uncertainty. Our  $\alpha$ Max- $B^3$  metric is simple to implement, has a solid theoretical foundation, and has many practical applications; making it an interesting evaluation metric in the clustering domain; and introducing a new supervised approach for assessing cluster quality with a particular emphasis on sub-groups in the context of coarse label-uncertainty.

### Take-Away

We introduced  $\alpha$ Max- $B^3$ , a new cluster evaluation metric that addresses issues with the conventional  $B^3$  cluster quality metric when dealing with coarse labels. It merges clusters into super-sets and applies a modified  $B^3$  metric to provide a fairer evaluation; accounting for coarse label uncertainty. The degree of uncertainty  $\alpha$  can be estimated automatically, or set manually.  $\alpha$ Max- $B^3$  is mathematically motivated; setting  $\alpha = 0$  yields the same results as  $B^3$ .

### Keywords

Clustering Results • Coarse Ground-Truth Labels • Supervised Evaluation

#### 10.5.1 Determining Representative Textual Labels for Clustering Accurate Sensor Data with Inexact Annotations

Finding the most fitting/representative textual annotation for a cluster group is especially challenging when there is a strong disparity in quality between sensor data and textual descriptions. While sensors are very reliable, textual data may be inconsistent, subjective, incorrect, or noisy. As a result, the quality of cluster representations in annotations may not correspond to that of the sensor data. This means that obtaining the most accurate annotation that aligns best with a cluster group is a key challenge. Particularly, even though sensor data is objective, different textual annotations can be used to express the same underlying sensor data. These annotations may encompass different information, leading to inconsistencies and varying levels of detail. There is, thus, uncertainty in the semantic alignment.

Uncertainty in the quality of textual annotations requires a cautious approach. While clustering sensor data offers objectivity and reliability, clustering textual data will likely not capture well the underlying real patterns of the sensor data (due to the [potential] misalignment). Combining both, i.e. concatenating the sensor embedding vector with the textual embedding vector, and clustering on the joint concatenated representation is *not* a good idea, since it increases the clustering dimension, introduces complexities, and we would be integrating low-quality text with high-quality sensor data; which can worsen the accuracy and reliability of clustering results. As a most extreme scenario, imagine a completely irrelevant or contextually unrelated textual comment. If we regard the vector embedding of this text as being random, we would essentially be adding random "noise" to a (perfectly) precise sensor embedding vector. Moreover, if the textual comment is even misrepresentative, wrong,

or the opposite, this will worsen the accuracy of embedding representation even further; even if the sensor embedding vector was perfectly descriptive (i.e. we would destroy or distort the good embedding representation vector).

The idea is to use clusters obtained from sensor data to achieve the following objectives: (1) evaluate the quality of textual annotations associated with these clusters, (2) determine a degree and ranking of "representativeness" for comments in regard to their sensor data or clusters, (3) determine the most representative textual annotation that best encapsulates the majority of instances within a cluster, (4) identify a suitable textual representation for a given cluster, and (5) identify if a comment is representative for its sensor data.

We used different representation scores (heuristics) and applied the harmonic mean. Scores can have high variability and we wanted to penalize extremely low score values (outliers). The harmonic mean is a valuable metric in such situations because it reduces the impact of individual high-value scores while amplifying the effect of low scores (i.e. outliers); making it suitable for scenarios where a balanced and robust average is required. In other words, outliers have a greater impact when they have smaller values (low scores), while their impact is minimized (i.e. less) when they have larger values (high scores) (Komić, 2011). Low scores are, thus, more severe and ranked worse.

This aligns with the challenges posed by the variability and diverse nature of the textual annotations in this context. The concept of a "representative" cluster annotation can be thought of: "Which of the comments best encapsulates the overall essence of a given cluster?". Hence, in particular, we want to filter out poor representatives clearly. A possible alternative to the harmonic mean would be using the geometric mean. The arithmetic mean, however, would not be a good choice here, since it does not penalize (too) low individual scores strong enough.

It is important to note that our focus here lies in identifying and retrieving the best *overall* representation. Specifically, we prioritize achieving an effective ranking between top instances to better ensure we really retrieve the best possible representation; rather than emphasizing having ranking precision in the lower rankings. In other words, our priority is to accurately rank the top instances, with little to no concern for maintaining an accurate ranking of bad instances; or informally, we do not care about the ranking precision of bad comments, we only care about identifying good comments. Therefore, it is unnecessary to achieve a perfect ranking across the *complete* data set; only across "top" comments (which we do via harmonic averaging).

### Take-Away

Aligning accurate sensor data with uncertain textual descriptions is important for effective clustering, learning joint embeddings, and many other related tasks. By identifying between representative and non-representative comments, we can do this without compromising the downstream quality; e.g. by filtering out the respective textual comments that do not accurately represent or align with the data.

### Keywords

Accurate Sensor Data • Inexact Text Data • Clustering • Ranking

## 10.6 Automated Textual Description Generation of Clusters

No pooling method was found to work best for all clusters, data sets, and decoding strategies. This is intuitive: the optimal pooling technique for a cluster depends on cluster-specific characteristics and the data distribution. However, still, basic pooling techniques already performed "surprisingly" well, especially when combined with a stochastic word sampling and sequence selection strategy. This is quite interesting since it suggests that, in large language models, the mean embedding representation of individual embeddings (from a cluster) is meaningful and quite representative of the overall cluster. However, we need to keep in mind that the cluster instances were already quite close in the embedding space in the first place; otherwise, they would not have been grouped together as a cluster.

Often, the *mean* is used as a cluster representation due to its simplicity and geometric interpretation, but it is sensitive to outliers and may not be suitable for all data distributions (such as non-convex sets). The *geometric median* (GM) may be a better central tendency measure for non-linear distributions, as it minimizes the sum of distances from data points and is less sensitive to outliers. Iterative methods like Weiszfeld's algorithm (Weiszfeld, 1936) can be used here to approximate the GM efficiently; the sum of distances to sample points is a convex function (Plastria, 2011; Minsker, 2015). Yet, the curse of dimensionality in high-dimensional spaces may make the GM an inaccurate cluster representation method due to the loss of meaning in low-density areas in geometric intra-cluster distances as data becomes increasingly scattered and sparse. Recall here the well-known phenomenon of "distance concentration" for high dimensions: the smallest distance between any two points becomes similar to the greatest distance between any two points.

*Random pooling* forwarded a random vector as the cluster representation through a decoder model and was used as a (lower) benchmark. *Max-pooling* was very often worse than random pooling; commonly producing repetitive token sequences or non-grammatical sentences (thus, performed poorly). This is also intuitive: since it always returns the maximum value of all cluster instances, it can be seen as the worst possible outlying point (lying even outside a cluster's convex hull). Therefore, it can easily produce an outlier vector outside the trained manifold space; leading so to skewed probabilities in the language model head. Moreover, max-pooling also easily results in semantic information loss as it only considers the maximum feature value; which is especially problematic if the maximum value is an outlier.

A generative model can produce an output, such as text, when given *any* input vector. By using random linear combinations of the hidden states of a cluster instance, we can thus generate multiple synthetic target captions within a particular cluster neighborhood. This allows us to explore and navigate the local manifold space within a given cluster region. When combined with CLIP weighting or a CLIP-based gradient-guided search, these synthetic texts can then also be used to additionally weigh those images proportionally stronger (during pooling) that exhibit the greatest similarity to the majority of the synthetic texts (i.e. local neighborhood).

During sequence generation, a decoder model predicts the next most probable token based on its contextual information, specifically, the preceding tokens. As a result, the model chooses tokens with the highest probabilities according to the 'language model', not the 'evaluation metric'. This is evident in Table 9.6 where the simple greedy decoding strategy outperformed the greedy beam-search. In other words, a model might generate an output sequence that it considers highly probable with a strong level of confidence, but this sequence could score low on a different evaluation metric. Hence, a population-based fitness function can be employed to

counteract this issue. First, multiple candidate sequences are generated that are all highly likely according to the model, and then the one with the highest score according to the metric is selected. In our experiments, this performed (as expected) the best. Selecting words (i.e. tokens) by *sampling* proportional to their token probabilities allows for a great variety in word selection; hence, a great variety in different candidate sequences.

On average, the contrastive approach achieved better performance overall (Tables 9.6 and 9.7). Interestingly, as previously mentioned, simple methods such as mean and median pooling already obtained very good and comparable scores, with only minor differences compared to the contrastive approach. In fact, almost all pooling and decoding strategies surpassed the mean intra-cluster performance. When combined with a population-based approach and a ranking function ( $k$ -sequence sampling), they even approached or exceeded the maximum-intra-cluster performance. In other words, we achieved scores that were comparable or even superior to the instance caption that is most similar to the entire cluster (i.e. to all candidate instance captions). However, instead of using a computationally intensive brute-force search, we generated such sequences using a generative model; via (cheap) aggregated manifold representations.

While simple averaging techniques can be effective, they may not result in a "most optimal" cluster representation. The ideal representation depends on various factors, such as a cluster's data distribution and semantic meaning in the latent manifold space. For example, static pooling methods (like the mean or median) solely calculate the statistical average based on raw numerical vector features; ignoring the semantic model manifold. Moreover, if non-convex clusters are formed through density-based techniques, the concept of the mean (or cluster centroid) can even fail to provide an accurate or correct cluster representation. Therefore, using simple static pooling methods may not adequately capture different data variations; i.e. given the many different possible data distributions within and between clusters. On the other hand, an adaptive approach such as maximizing contrastive Language-Image scores *does* consider the hidden manifold space, which can result in better and more fitting representations; hence, better CCs (we achieved slightly higher scores).

#### Take-Away

We noticed that combining latent representations of neighboring cluster instances on a manifold yields a meaningful hidden cluster state; which can be used to generate representative descriptions for individual clusters. Our experiments further indicate that a dynamic pooling strategy can produce slightly better descriptions than a fixed static pooling approach. However, simple pooling methods were already quite effective (and faster to compute).

#### Keywords

Embedding Aggregation • Latent Representation • Manifold • Cluster State

## 10.7 Misaligned Output Calibration for Out-of-Distribution Detection

**Please note:** This contribution has been accepted for publication after peer review in Guimerà Cuevas and Schmid, 2024a and has been presented in the conference

proceedings.

Naively, one approach to outlier detection is to train one-versus-rest classifiers for all classes and use the difference between 1 and the sum of probabilities as an outlier measure; similar to Membership Loss (Perera and Patel, 2019). This assumes that if an example is an outlier, it will not fit into any class. However, such approach of using one-versus-rest classifiers and comparing probabilities may not really be sufficient or accurate for detecting outliers: e.g., assuming outliers do not fit into any class may be oversimplified; imbalanced data can bias detection towards majority classes; complex decision boundaries may hide outliers near class boundaries; and the reliability of probability scores as an outlier measure varies across classifiers and datasets, and they can be miscalibrated and fail to reflect the true certainty of a model's predictions. Also, it is not a post-hoc method (but ad-hoc) and requires "modifying" the training process, which can be computationally expensive for large data sets or models.

In our method, we identify outliers by looking at examples where the one-versus-rest probabilities deviate significantly from the expected values (i.e. calibration). In other words, outliers are instances where a model's prediction is not aligned with the calibration. If a prediction is over- or underconfident and consistently misaligned with the calibration across all/most classes, it can be considered anomalous and outside the distribution of the training set (out-of-distribution [OOD]). The more an instance deviates from the in-distribution calibration, the more likely it is an OOD sample. By using the L2 norm on the miscalibration vector, examples are more likely to be classified as outliers the less uniformly the one-versus-rest probabilities are distributed. This is helpful in detecting not only OOD samples but, additionally, also instances where the model is especially uncertain or confidently wrong; e.g., a uniform distribution of probabilities suggests that the model recognizes the example does not belong to any class. Therefore, the L2 norm introduces a weighting that penalizes instances where the model is overconfident in predicting a specific class. Generally speaking, it is much worse to have outliers incorrectly classified as inliers, compared to having outliers that the model correctly "sees" as outliers. Thus, one may argue that outliers that result in overconfident predictions are worse; and should, hence, be penalized more.

As previously mentioned, there is no perfect solution or universal approach to detecting outliers, just like there is no one-size-fits-all method for clustering. Each algorithm focuses on different characteristics and defines outliers in different ways. Outlier scores (from different algorithms) may come in different distributions, sensitivities, and scales. To compare or combine the results from different algorithms, we need to ensure *unified scores* (Kriegel et al., 2011). Depending on the specific task at hand, we may also need to prioritize OOD precision, recall, or find a balance between the two (when comparing/combining multiple outlier scores).

Overall, in the context of DNNs, we can use calibration misalignments to improve the detection and recognition of OOD samples. Our approach works on the DNN's prediction outputs, without requiring the integration of OOD calibration into the model training process. This is advantageous because, otherwise, complete (or partial) retraining can be costly and may lead to loss, noise, or damage to previous knowledge; especially if having to start (i.e. retrain) completely from scratch.

**Take-Away**

Different outlier detection algorithms have different characteristics, and there is no one-size-fits-all solution. Calibration misalignments can be leveraged to detect out-of-distribution samples; without requiring model retraining.

**Keywords**

Outlier Detection • One-vs-Rest Classifiers • Calibration Misalignments

## 10.8 Representative Sampling in Data Streams

We have already underscored the importance of obtaining "representative" data samples from a potentially infinite data stream; and presented a technique for maintaining samples within a fixed range of quantiles. For trajectory sampling, where making predictions based on both past instances and current ones is of interest, we have suggested combining random (but representative) quantile samples with the most recent samples to make predictions. Including the top newest samples ensures that we are always guaranteed to take the newest samples into consideration while including the quantile sample provides a past (i.e. earlier) context. We have demonstrated several mathematical properties of our quantile sampling method, which define the characteristics of what can be deemed as "representative samples". These properties e.g. include having an equal probability of being included, being removed from the reservoir when they expire, etc.

Quantile sampling poses a challenge in removing expired elements (and replacing them) at the right time due to the continuously increasing quantile window size, especially in the initial phase of data streaming when it is more unstable than towards the end. As new data points arrive, the size of the data stream grows, resulting in a larger quantile window. Consequently, older data points eventually fall outside of the window and are considered expired. If these expired elements are removed too soon, valuable information from more recent data points may be lost, which can negatively impact the representativeness of the sample. Conversely, if expired elements are removed too late, the sample may become outdated and fail to accurately represent the current/latest state of the data stream. Notice here how, as a consequence of limited memory (and the [possibly] infinite data stream), we are unable to store every element in the stream: we have to selectively sample items in real-time and determine which to remove (permanently), and which to retain.

In the early stages of the data stream, the size of the quantile window increases and changes more rapidly; hence, is more unstable. Its size is  $\lceil N\phi \rceil$ , where  $N$  is the total number of data points and  $\phi$  represents the desired quantile window size. E.g., consider  $\phi = 10\%$  and three scenarios:  $N = 100$ ,  $N = 100,000$ , and  $N = 1,000,000$ . In the first case, the window size increases after every 10 new data points. In the second case, it increases after every 10,000 points. And in the third case, it increases after every 100,000 points. Obviously, the sampling process is more unstable and challenging in the beginning because the quantile window size changes rapidly. However, as the data stream progresses, the window size increases at a slower pace, leading to a more stable sampling process. Thus, if the data stream or trajectory is small, there may be issues with using the quantile sampling method; and a different sampling technique might be more appropriate. In fact, if the data stream is very small, there may be *no* necessity for quantile sampling at all and we could e.g.

choose to store or handle the full sequence directly. Ultimately, quantile sampling is a Big Data algorithm; and not meant (i.e. designed) for small sequences.

#### Take-Away

Representative sampling is relevant in infinite data streams, where predictions can be made using a combination of random quantile samples and recent data. "Representativeness" can be ensured by mathematical properties. Removing and replacing expired elements is important. The quantile window size grows, particularly in the early stages of the data stream.

#### Keywords

Representative Data Samples • Infinite Data Streams • Trajectory Sampling

## 10.9 Clustering on Latent Manifold Structures

We have explored clustering directly on the manifold space, rather than on the geometric or Euclidean space. Such an approach relies on the assumption that similar or related objects are effectively encoded on the same manifold. Due to the complex geometries of latent manifolds, identifying neighboring instances on the manifold is not as straightforward as it is using the Euclidean distance. To cluster correctly on the manifold, it is essential to define a metric for manifold closeness. However, DNN manifolds are typically *not* smooth and noisy, sharp, and complex; making manifold clustering considerably more challenging.

Unlike Euclidean-based clustering, manifold clustering requires access to the model since without the underlying manifold, it cannot be performed. Accessing the model itself can already be a significant obstacle (or impossible) in certain use cases. This is a key limitation for such manifold clustering methods which require white-box access to the model's latent manifold function (as in our proposed Taylor Approximation error). In other words, in situations where only limited knowledge about the model is available (black-box access), these methods cannot (really) be applied. Additionally, manifold clustering is more computationally demanding than merely computing Euclidean distances. Nevertheless, we consider this a characteristic of manifold methods rather than a limitation. After all, manifold-based algorithms (per definition) rely on and require a manifold; which, in turn, provides much more meaningful and contextual information than the Euclidean space does. In fact, the Euclidean embedding space is merely a projection of the manifold space.

In particular, manifold clustering can help uncover hidden (latent) patterns, clusters, or subgroups within the data that may not be easily identifiable using traditional clustering methods. It allows us to leverage the knowledge encoded in the model and directly exploit manifold properties.

Overall, we believe manifold clustering provides an interesting perspective for exploring and clustering data by directly considering the underlying manifold; which, again, can be considered of high value since it encodes the learned semantics and logic of a particular trained DNN for determining embeddings and internal representations (mapping input data to latent hidden states); as opposed to Euclidean-based clustering.

**Take-Away**

Manifold clustering groups data based on the underlying manifold space. It can uncover hidden latent patterns (i.e. clusters) that traditional methods may miss; but it requires access to the model and is computationally intensive.

**Keywords**

Manifold Clustering • Latent Hidden Patterns • White-Box Model Access



## Chapter 11

# Summary & Conclusions

In this thesis, we incorporated "Take-Away" boxes at the end of each (main) section to emphasize and highlight important key insights.

This chapter offers comprehensive summaries and conclusions regarding the main topics and central ideas discussed throughout the thesis. Specifically, in this thesis, we focused on our three key domains: (1) neural architecture and data representation, (2) feature pre-processing and representation learning, and (3) knowledge transfer, outlier detection, and sampling. For specific details, we refer to the corresponding sections in the thesis. Overall, the thesis covered the following areas:

### 1. NEURAL ARCHITECTURE AND DATA REPRESENTATION

- **Complex-Valued Networks:** Architecture based on complex numbers.
- **Hierarchical Database Encoding:** Efficient processing via one-to-many relationship database encoding.
- **Synthetic Data Augmentation:** Synthetic samples for hierarchical databases.
- **Contrastive Representation Learning:** Framework for representative embeddings in tree structures.
- **Concatenated Representation Learning:** Unifying multiple data views.

### 2. FEATURE PRE-PROCESSING AND REPRESENTATION LEARNING

- **Advanced Feature Normalization:** Adaptive feature processing technique for improving training stability and outlier robustness.
- **Representation Learning and Manifold Clustering:** Dimensionality reduction of complex high-dimensional data and clustering on manifolds.
- **Clustering Evaluation Metric:** Supervised cluster quality evaluation considering coarse label-uncertainties.
- **Textual Description Generation:** Generating textual cluster descriptions.
- **Representative Textual Labels:** Determining alignment between sensor and textual data.

### 3. KNOWLEDGE TRANSFER, OUTLIER DETECTION, AND SAMPLING

- **Transfer Learning via Knowledge Distillation:** Leveraging pre-trained model insights.
- **Out-of-Distribution Detection:** Outlier detection via model calibration mismatches.
- **Representative Sampling:** Stream sampling within quantile windows.

## 11.1 Neural Architecture and Data Representation

### 11.1.1 Complex-valued based Neural Networks

**Notice:** Our work and findings on complex-valued NNs (presented in this thesis) have been published in Guimerà Cuevas, Phan, and Schmid, 2023. It was inspired and builds upon our previous research (Guimerà Cuevas and Phan, 2021), but differs significantly as it has been substantially improved and modified (both the complex architecture and its mathematical framework); thus, constitutes novel contributions. In particular, this novel contribution has been accepted for publication, after peer review. The Version of Record is available online at: [https://link.springer.com/chapter/10.1007/978-3-031-33374-3\\_28](https://link.springer.com/chapter/10.1007/978-3-031-33374-3_28). Use of this Version is subject to the publisher's Manuscript terms of use. Therefore, please refer to Guimerà Cuevas, Phan, and Schmid, 2023 for specific details.

An adaptive bi-nonlinear layer architecture is introduced, where model weights are constrained to lie on the unit circle. This constraint prevents the elimination of neural connections by setting weights to zero and ensures that no single weight becomes disproportionately dominant. In this model, weights induce phase rotations, while the input governs amplitude scaling. The architecture retains the familiar topology of traditional neural networks, with a similar number of trainable parameters; potentially more when considering bias and angular weights. Training is accomplished using gradient descent and backpropagation, with the model showing robustness at high learning rates. It effectively completed various tasks across different architectures, able to demonstrate greater expressiveness than real-valued networks. However, the forward pass requires double the number of matrix multiplications - one for the real component and one for the imaginary component (7.1).

### 11.1.2 Hierarchical Heterogeneous Graph Neural Networks

A method was presented for applying DL on HDBs with recursive one-to-many relations which computes embedded representations for its entities and allows downstream tasks (e.g. classification or clustering) to be performed. It encodes DB relations as edges and DB tables as nodes; but the proposed encoding and model scheme is not restricted to HDBs because it can be applied to any data that can be turned into a hierarchical tree-like structure with loop-free connections; the sole constraint is that node features under a common parent be equivalent (or a virtual parent node has to be included), but otherwise, they may well be different across other nodes (e.g. at different hierarchy levels). Node entities (i.e. DB tables) containing different features are - in contrast to standard graph neural networks - directly supported. The proposed HDB-model architecture has also key advantages over naive transformer-based encoding approaches which (forcefully) encode the HDB in textual form; particularly in terms of the ability to process numeric features directly and avoid truncating long encoded token sequences. Experiments on both real-world and synthetic data demonstrated the effectiveness of the proposed method. We found that performance in purely supervised classification tasks was improved by adding residual connections, while in semi-supervised tasks where identity and information-preservation are important, the use of a Kernel embedding improved performance (7.2).

### 11.1.3 Synthetic Data Augmentation for Tree Structures

We explored tree-data augmentation, a method to expand datasets by generating synthetic instances from hierarchical graphs. We employed a "merge & sample" strategy: combining graphs to form unified trees and extracting random sub-graphs. However, we observed that this can lead to biased augmentation in imbalanced datasets; for uniform distribution-based sampling. To address this issue, we proposed a solution that involves conditional sampling based on graph rarity, measured by Jaccard-Coefficient (JC) similarity scores; but computing the JC matrix for large datasets can pose computational challenges. To mitigate the sample bias, we used an inversely proportional probability distribution over the summation of the similarity scores to increase the likelihood of sampling uncommon tree structures (7.4).

### 11.1.4 Contrastive Representation Learning & Loss-Weighting

A contrastive loss is crucial for guiding models to distinguish between similar and dissimilar samples, helping in managing intra-class variations, refining similarity measurements, and enhancing resilience against data noise or outliers; to ultimately lead to better generalization.

Our main goal in this thesis was to generate meaningful representations of (hierarchical) tree structures. To accomplish this, we incorporated various loss functions; particularly, similarity-contrastive losses for model training (alongside others like the classification loss). These loss functions were specifically designed to encourage the clustering of similar trees closer together while pushing apart dissimilar trees within the embedding space. The similarity loss ensured that comparable trees yielded similar embeddings, while the contrastive losses emphasized differences between similar and dissimilar embeddings. The other losses served different purposes, e.g. the classification loss shaped graph embeddings to reduce label-induced representation errors.

To create a single, unified objective function, we combined the losses via careful weighting. However, the CoV-Weighting method, although dynamically assigning weights during optimization, treats all losses initially as equally important. To address this, our proposed modification was to integrate priority weights into the CoV-Weighting to better prioritize specific losses during training (7.5).

### 11.1.5 Concatenated Representation Learning & Separate Data Training

The concatenated representation learning approach handles multiple different data views within a unified framework; merging representative embedding vectors into a single concatenated vector. This enables capturing patterns across the different views, allowing for flexible and scalable training without the need to retrain entire models when individual data views change or new ones are introduced. Hence, instead of training a large single model, we adopted the approach of training each data view separately and, afterward, jointly in combination (over the respective individual embedding representations). This offers the advantage of enabling parallel training and effectively avoids potential performance bottlenecks arising from variations in the sizes and complexities of different data views. Moreover, it enables the implementation of memory-efficient strategies for handling duplicate instances, facilitating similarity search, and enhancing graph batching; optimizations were applied separately to each data view, resulting in more efficient training (7.6).

## 11.2 Feature Pre-Processing & Representation Learning

### 11.2.1 Robust Non-linear Normalization of Heterogeneous Feature Distributions with Adaptive Tanh-Estimators

**Please note:** This contribution has been accepted for publication, after peer review in Guimerà Cuevas and Schmid, 2024b. The Version of Record is available online.

Correctly pre-processing features is of great importance for achieving successful and effective ML models. We focused on the Tanh-Normalization method and highlighted a key limitation (and its consequences): the fixed static global spread value parameter can cause slow training convergence and poor model performance, and may require time-consuming parameter tuning if optimized manually. Clearly, this is not optimal or efficient. To address this issue, we proposed an adaptive version of the Tanh-Normalization method that automatically determines the optimal spread value (for each feature) by minimizing the Wasserstein distance to a desired target distribution (the ideal output distribution the normalization should follow). This not only improves training convergence but also eliminates the need for manual parameter tuning. Our proposed method maintains outlier resilience, efficiency, and simplicity while ensuring an effective distribution of probability mass. In particular, this is achieved on a per-feature basis (i.e., locally), as opposed to using a fixed global spread value. We repeatedly emphasized the importance of proper feature pre-processing and its impact on ML models (7.3).

### 11.2.2 Completeness and Uncertainty in Cluster Evaluation

**Notice:** Our work and findings on  $\alpha$ Max-B-CUBED (presented in this thesis) have been publicly released on OpenReview-Venue (Guimerà Cuevas and Schmid, n.d.).

We have identified a concern regarding the widely used  $B^3$  cluster quality metric when coarse labels are involved, which can lead to unfair and misleading evaluations. To address this issue, a new metric called  $\alpha$ Max- $B^3$  has been proposed. This solution modifies the evaluation of the standard  $B^3$  method by adapting to sub-group uncertainty in ground-truth labels and can be generalized to accommodate imbalanced data sets. The proposed evaluation method merges clusters into larger groups called super-sets and evaluates them using a modified  $B^3$  based metric that applies a weighting factor to control the contribution of the super-sets. Unlike the standard  $B^3$  technique,  $\alpha$ Max- $B^3$  can produce more robust and fair results and adapt to label uncertainty. This uncertainty is controlled by an  $\alpha$  parameter and setting it to zero yields the same results as the standard  $B^3$  metric. Our solution is easy to implement, has a solid theoretical foundation, and has many practical applications; making it an attractive evaluation metric in the field of clustering (7.9).

### 11.2.3 Representation Learning and Manifold Clustering

We proposed a new manifold clustering approach that considers the underlying manifold space directly via Taylor approximation errors; instead of using traditional clustering metrics like the Euclidean distance. The main assumption and idea behind this method is that similar items share a common manifold; or reside within smooth manifold transitions. Or in other words, this means that similar items have latent encoding representations that are drawn from the same manifold distribution. The key implication then is that, if similar items have similar manifold encodings, then by traversing the local manifold shape, we should, hence, find similar items. This we can use and exploit for clustering. However, there are some challenges in

defining manifold metrics, such as dealing with bumpiness or imperfections within manifold spaces, and the need for model access. Despite these limitations, manifold clustering offers a unique and different perspective in identifying data patterns and clusters than other conventional non-manifold-based methods or metrics; the key idea of manifold clustering is to leverage the encoded knowledge within models (i.e. in the manifold space, instead of e.g. in the Euclidean space). Overall, the two fundamental challenges are: (1) determining a representative, meaningful, and correct manifold space, and (2) identifying this manifold space and establishing a reliable metric to cluster and detect neighboring manifold instances (7.7).

#### 11.2.4 Manifold Aggregations of Latent Spaces for Generating Textual Descriptions of Clusters

We found that we can effectively generate descriptions of clusters by combining the representations of nearby instances on the manifold of large generative (language) models. This can be achieved by dynamic pooling methods on the latent states of the neighboring cluster instances, but also even using simple statistical or geometrical manifold aggregation strategies. We can further improve the quality of the generated descriptions by combining it with a population-based decoding sampling strategy; ranked by a score function. We introduced two dynamic pooling methods that use contrastive Language-Image scoring models. One method employs a gradient-based search towards a unified manifold, while the other uses contrastive scores for weighted mean pooling. On average, both of these methods outperform the naive static pooling methods slightly.

Two key insights of this are: (1) aggregating latent representations of neighboring instances on a manifold forms an effective hidden cluster state for encoding or extrapolating cluster knowledge, and (2) although surprisingly effective, the optimal hidden cluster state representation is *not* achieved through a simple static (e.g. mean) pooling; it relies on various factors like the underlying manifold structure and the distribution of instances within the cluster in the manifold space. Here, our results showed that a dynamic pooling method, e.g. maximizing the contrastive Language-Image score of the unified representation across all instance descriptions in a cluster, can yield better cluster states (7.10).

#### 11.2.5 Determining Representative Textual Labels for Clustering Accurate Sensor Data with Inexact Annotations

Identifying representative textual annotations for cluster groups that are derived from sensor data is a challenge because of differences in data quality. While sensor data is usually precise, textual descriptions can vary greatly in terms of consistency, subjectivity, and accuracy. This makes it difficult to align textual annotations with sensor readings because different textual descriptions may relate to the same sensor data in different ways. However, integrating low-quality text with high-quality sensor data for clustering can reduce accuracy and reliability. Our objective was to assess the quality of textual annotations, identify representative comments for clusters of sensor data, and find appropriate textual representations that do *not* compromise the quality of sensor data when combined. Therefore, we introduced an intuitive heuristic to rank textual annotations by their (cluster) representation quality (7.12).

## 11.3 Knowledge Transfer, Outlier Detection, and Sampling

### 11.3.1 Transfer Learning via Knowledge Distillation

We have outlined a systematic approach for transferring knowledge across different models and input domains. To begin with, we distilled insights from an interpretable model into a stronger intermediate model, and then reversed the process back again to create a (new) final interpretable model. However, the rules of this new model were now based entirely on a different dataset (than the initial model). We highlighted the challenges involved in extracting such new interpretable rules (drawn from the new dataset), such as preventing information loss and biases, and emphasized the importance of the intermediate model's quality for achieving good final rules. Overall, our approach adapts/transfers knowledge from an old dataset to a new dataset; and is so, suitable for changing input domains (7.13).

### 11.3.2 Misaligned Calibration for Out-of-Distribution Detection

**Please note:** This contribution has been accepted for publication after peer review in Guimerà Cuevas and Schmid, 2024a and has been presented in the conference proceedings.

Detecting OOD examples is a difficult task, and different algorithms have varying levels of success depending on the data distribution and outlier types. We proposed a post-hoc method for detecting OOD examples in DNNs by measuring multi-class output calibration misalignment. Our approach is based on the assumption that OOD examples are more likely to exhibit anomalous behavior, which results in a discrepancy in calibration alignment (against in-distribution examples). Empirical results demonstrate that our method is effective for DNNs and can outperform or match the performance of many common outlier detection algorithms (7.14).

### 11.3.3 Representative Sampling in Data Streams

Data streams, time series, and trajectories are important concepts in data analysis that present unique challenges for ML algorithms. These challenges arise due to the typically *infinite* and continuously changing nature of input streams, the inconsistency in data arrival rates and quantities, and due to variations in data quality when collected from different sources. To address computational and data storage (i.e. memory) limitations, representative data stream sampling is crucial. Yet, achieving a "representative" sample can be difficult; also due to challenges like concept drift, imbalanced data, potential biases, real-time processing, etc. Ensuring a *representative* sample, however, is important for accurate analysis, processing, and avoiding incorrect conclusions. Other limitations such as noise and temporal (or e.g. geographical) data relevance can further complicate the analysis of data streams; particularly in time series and trajectories. In this thesis, we have proposed and analyzed an online sampling method based on Reservoir Sampling. This method allows us to maintain a representative temporal sample within a dynamic quantile window, without prior knowledge of the stream's length. Thus, it can handle infinite streams. By preserving samples for different quantiles of the data distribution, we can determine the focal point of the sample for different quantile windows. This can also be combined to additionally include the most recent samples. The term "representative" is defined based on the specific mathematical properties of the sampling algorithm (7.16).

## 11.4 Concluding Words

This thesis covered various ML topics and challenges: providing analysis, literature overview, insights, and new solutions. The main focus of the thesis was on Representation Learning for recursive hierarchical databases; the primary challenge was to create a model architecture to allow for effective ML on these complex data structures (we proposed a recursive DL design pattern architecture). We emphasized the importance of obtaining meaningful and valuable data representations to enable different downstream tasks; e.g. classification, similarity search, outlier detection, etc. Respectively, we proposed specific objective losses for learning such effective embedding representations. Overall, we introduced several novel ML methodologies and improvements that particularly focused on different key areas: "neural architecture and data representation", "feature pre-processing and representation learning", and "knowledge transfer, outlier detection, and sampling". The proposed solutions were designed to improve the effectiveness and robustness of automated strategies, addressing prevalent AI challenges and applications, with a particular focus on industrial use cases.



## Appendix A

# Out-of-Distribution Detection

**Please note:** This contribution has been accepted for publication after peer review in Guimerà Cuevas and Schmid, 2024a and has been presented in the conference proceedings.

### A.1 Proofs

Consider a probability vector  $\vec{p}$  of dimension  $k$ , where the elements sum up to one, expressed as  $\mathbb{1}^T \vec{p} = 1$ ; where  $\mathbb{1}$  represents a vector or sequence of ones. We show that  $\mathbb{1}^T(\mathbb{1} - \vec{p}) = k - 1$ . This is evident by:

$$\mathbb{1}^T(\mathbb{1} - \vec{p}) = \sum_{i=1}^k (1 - p_i) = \sum_{i=1}^k 1 - \sum_{i=1}^k p_i = \sum_{i=1}^k 1 - \sum_{i=1}^k p_i = k - 1 \quad (\text{A.1})$$

Now consider pairs-wise tuples  $p_{n=2}^{[ij]} := p_i + p_j$ . Since we are dealing with pairs of two, we have  $\binom{k}{2}$  pairs; forming a new vector  $\vec{p}_{n=2}$  of respective dimension. Then:

$$\mathbb{1}^T(\mathbb{1} - \vec{p}_{n=2}) = \sum_{i < j} (1 - p_{n=2}^{[ij]}) = \sum_{i < j} 1 - \sum_{i < j} p_{n=2}^{[ij]} = \binom{k}{2} - \sum_{i < j} p_{n=2}^{[ij]} \quad (\text{A.2})$$

To further analyze the expression  $\binom{k}{2} - \sum_{i < j} p_{n=2}^{[ij]}$ , we can note that  $\sum_{i < j} p_{n=2}^{[ij]}$  is equivalent to summing the probabilities of all possible pairs of elements, and since:

$$\sum_{i < j} p_{n=2}^{[ij]} = \sum_{i < j} (p_i + p_j) = \sum_{i=1}^k \sum_{j=1+i}^k (p_i + p_j) = \sum_{i=1}^k (k-1)p_i = (k-1) \sum_{i=1}^k p_i = (k-1) \cdot 1 \quad (\text{A.3})$$

we know a specific item occurs  $k - 1$  times when considering all combinations (i.e. it is paired exactly once with every other item). Thus, we have:

$$\mathbb{1}^T(\mathbb{1} - \vec{p}_{n=2}) = \binom{k}{2} - (k-1) \quad (\text{A.4})$$

Now, consider tuples of size  $n \in \mathbb{N}^* < k$ , i.e.  $p_n^{[i_1, i_2, \dots, i_n]} := p_{i_1} + p_{i_2} + \dots + p_{i_n}$ . Hence, we have  $\binom{k}{n}$  many tuples; forming a new vector  $\vec{p}_n$  of respective dimension:

$$\begin{aligned}\mathbb{1}^T(\mathbb{1} - \vec{p}_n) &= \sum_{i_1 < i_2 < \dots < i_n} (1 - p_n^{[i_1, i_2, \dots, i_n]}) = \sum_{i_1 < i_2 < \dots < i_n} 1 - \sum_{i_1 < i_2 < \dots < i_n} p_n^{[i_1, i_2, \dots, i_n]} \\ &= \binom{k}{n} - \sum_{i_1 < i_2 < \dots < i_n} p_n^{[i_1, i_2, \dots, i_n]}\end{aligned}\tag{A.5}$$

Again,  $\sum_{i_1 < i_2 < \dots < i_n} p_n^{[i_1, i_2, \dots, i_n]}$  is equivalent to summing the probabilities of all possible tuples of  $n$  elements. Analogously to before, we know that:

$$\begin{aligned}\sum_{i_1 < i_2 < \dots < i_n} p_n^{[i_1, i_2, \dots, i_n]} &= \sum_{i_1 < i_2 < \dots < i_n} (p_{i_1} + p_{i_2} + \dots + p_{i_n}) \\ &= \sum_{i=1}^k \binom{k-1}{n-1} p_i = \binom{k-1}{n-1} \sum_{i=1}^k p_i = \binom{k-1}{n-1} \cdot 1\end{aligned}\tag{A.6}$$

since a specific item occurs  $\binom{k-1}{n-1}$  times when considering all combinations. Thus:

$$\mathbb{1}^T(\mathbb{1} - \vec{p}_n) = \binom{k}{n} - \binom{k-1}{n-1}\tag{A.7}$$

In other words, the expression  $\binom{k}{n} - \binom{k-1}{n-1}$  is the sum of combinations of probability tuples  $\mathbb{1}^T(\mathbb{1} - \vec{p}_n)$ . This generalizes the original proof established for pairs of tuples with a size of two; i.e. it completes the proof for tuples of any size  $n$ . For example, when  $n = 2$ , we have  $\binom{k}{2} - \binom{k-1}{1} = \binom{k}{2} - (k-1) = \mathbb{1}^T(\mathbb{1} - \vec{p}_{n=2})$ , consistent with Equation A.4. Moreover, this demonstrates that, given fixed  $n, k$ , the sum remains a *constant* factor regardless of the distribution of the probability vector  $\vec{p}$ ; and so, only depends on  $n$  and  $k$  (not  $\vec{p}$ ).

## Appendix B

# Clustering

### B.1 Unsupervised Clustering of Hierarchical Databases

In Figure B.1, the unsupervised deep representations of the projected embeddings are displayed. Additionally, the figure shows instances that are located in close proximity to each other in the embedding space. The results are complementary (but independent) to the ones shown in Chapter 9.4.1.

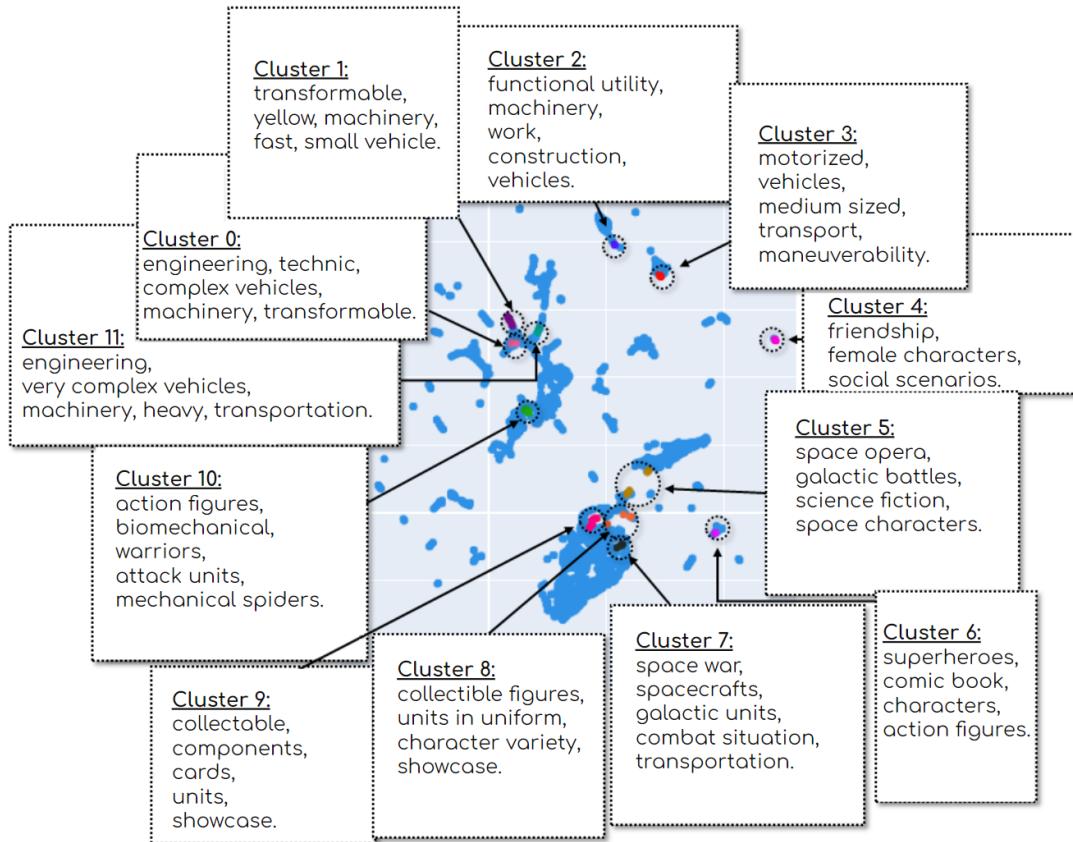


FIGURE B.1: The embedding projection and the closest points of random instances within some clusters are visualized. The images of the instances are from the data set (Rebrickable®, 2022). The results are independent of the ones shown in Figure 9.4.1; it depicts a completely different training run (on the same data set).

## B.2 Proofs

**Notice:** Our work and findings on  $\alpha$ Max-B-CUBED (presented in this thesis) have been publicly released on OpenReview-Venue (Guimerà Cuevas and Schmid, n.d.).

### B.2.1 Theorem 1

Let  $C_i, C_j$  be disjoint clusters of distinct elements. Theorem 1 is to be proven:

$$\mathbb{E}[P_{[y]}(C_i) \wedge P_{[y]}(C_j)] \geq P_{[y]}(C_i \cup C_j)$$

where  $P_{[y]}(C_j)$  denotes the precision scores of label  $y$  on elements in a cluster  $C_j$ . Let  $|C_i|, |C_j| \in \mathbb{N}$  denote the total number of elements in  $C_i, C_j$  and  $|C_i|_y, |C_j|_y \in \mathbb{N}$  respectively the number of elements in  $C_i, C_j$  with label  $y$ .

We know  $0 \leq |C_j|_y \leq |C_j| \wedge 0 \leq |C_i|_y \leq |C_i|$ . Therefore:

$$\begin{aligned} & \mathbb{E}[P_{[y]}(C_i) \wedge P_{[y]}(C_j)] \geq P_{[y]}(C_i \cup C_j) \\ \iff & \frac{|C_i|_y P_y(C_i) + |C_j|_y P_y(C_j)}{|C_i|_y + |C_j|_y} \geq P_y(C_i \cup C_j) \\ \iff & \frac{|C_i|_y \frac{|C_i|_y}{|C_i|} + |C_j|_y \frac{|C_j|_y}{|C_j|}}{|C_i|_y + |C_j|_y} \geq \frac{|C_i|_y + |C_j|_y}{|C_i| + |C_j|} \\ \iff & \frac{\frac{|C_i|_y^2}{|C_i|} + \frac{|C_j|_y^2}{|C_j|}}{|C_i|_y + |C_j|_y} - \frac{|C_i|_y + |C_j|_y}{|C_i| + |C_j|} \geq 0 \\ \iff & \frac{|C_j||C_i|_y^2 + |C_i||C_j|_y^2}{|C_i||C_j|(|C_i|_y + |C_j|_y)} - \frac{|C_i|_y + |C_j|_y}{|C_i| + |C_j|} \geq 0 \\ \iff & \frac{(|C_i||C_j|_y - |C_j||C_i|_y)^2}{|C_i||C_j|(|C_i|_y + |C_j|_y)(|C_i| + |C_j|)} \geq 0 \\ \iff & (|C_i||C_j|_y - |C_j||C_i|_y)^2 \geq 0 \iff \top \end{aligned} \tag{B.1}$$

### B.2.2 Proposition 1

We prove Proposition 1, which states that:

$$\mathbb{E}[P_{[y]}(C_i) \wedge P_{[y]}(C_j)] = P_{[y]}(C_i \cup C_j) \iff P_{[y]}(C_i) = P_{[y]}(C_j)$$

where  $P_{[y]}(C_j)$  are the precision scores of label  $y$  on elements in a cluster  $C_j$ . Let  $0 \leq |C_j|_y \leq |C_j| \wedge 0 \leq |C_i|_y \leq |C_i|$  be as in Proof B.2.1; we know the inequality:

$$\mathbb{E}[P_{[y]}(C_i) \wedge P_{[y]}(C_j)] \geq P_{[y]}(C_i \cup C_j) \iff (|C_i|_y |C_j| - |C_j|_y |C_i|)^2 \geq 0 \tag{B.2}$$

and so:

$$\begin{aligned}
& \mathbb{E}[P_{[y]}(C_i) \wedge P_{[y]}(C_j)] = P_{[y]}(C_i \cup C_j) \\
& \iff (|C_i|_y |C_j| - |C_j|_y |C_i|)^2 = 0 \\
& \iff |C_i|_y |C_j| - |C_j|_y |C_i| = 0 \\
& \iff \frac{|C_i|_y}{|C_i|} = \frac{|C_j|_y}{|C_j|} \\
& \iff P_{[y]}(C_i) = P_{[y]}(C_j)
\end{aligned} \tag{B.3}$$

### B.2.3 Corollary 1

We prove Corollary 1, which states that:

$$\mathbb{E}[P_{[y]}(C_i) \wedge P_{[y]}(C_j)] > P_{[y]}(C_i \cup C_j) \iff P_{[y]}(C_i) \neq P_{[y]}(C_j)$$

According to Theorem 1,  $\mathbb{E}[P_{[y]}(C_i) \wedge P_{[y]}(C_j)] \geq P_{[y]}(C_i \cup C_j)$  always holds. Therefore, if  $P_{[y]}(C_i) \neq P_{[y]}(C_j)$ , we apply Proposition 1 to conclude  $\mathbb{E}[P_{[y]}(C_i) \wedge P_{[y]}(C_j)] \neq P_{[y]}(C_i \cup C_j)$ . Consequently, it must follow that  $\mathbb{E}[P_{[y]}(C_i) \wedge P_{[y]}(C_j)] > P_{[y]}(C_i \cup C_j)$ .

### B.2.4 Theorem 2

Let  $C_i, C_j$  be disjoint clusters of distinct elements. Theorem 2 is to be proven:

$$\mathbb{E}[R_{[y]}(C_i) \wedge R_{[y]}(C_j)] \leq R_{[y]}(C_i \cup C_j)$$

where  $R_{[y]}(C_j)$  denotes the  $B^3$  recall scores of label  $y$  on elements in a cluster  $C_j$ . Let  $|y| \in \mathbb{N}$  denote the total number of elements with label  $y$  in all clusters, i.e. in the entire data set, and  $|C_i|_y, |C_j|_y \in \mathbb{N}$  respectively the number of elements in clusters  $C_i, C_j$  with label  $y$ . Then:

$$\begin{aligned}
& \mathbb{E}[R_{[y]}(C_i) \wedge R_{[y]}(C_j)] \leq R_{[y]}(C_i \cup C_j) \\
& \iff \frac{|C_i|_y R_{[y]}(C_i) + |C_j|_y R_{[y]}(C_j)}{|C_i|_y + |C_j|_y} \leq R_{[y]}(C_i \cup C_j) \\
& \iff \frac{(|C_i|_y \frac{|C_i|_y}{|y|}) + (|C_j|_y \frac{|C_j|_y}{|y|})}{|C_i|_y + |C_j|_y} \leq \frac{|C_i|_y + |C_j|_y}{|y|} \\
& \iff 0 \leq \frac{|C_i|_y + |C_j|_y}{|y|} - \frac{|C_i|_y^2 + |C_j|_y^2}{|y|(|C_i|_y + |C_j|_y)} \\
& \iff 0 \leq \frac{(|C_i|_y + |C_j|_y)^2 - |C_i|_y^2 - |C_j|_y^2}{|y|(|C_i|_y + |C_j|_y)} \\
& \iff 0 \leq \frac{2|C_i|_y |C_j|_y}{|y|(|C_i|_y + |C_j|_y)} \\
& \iff 0 \leq |C_i|_y |C_j|_y \iff \top
\end{aligned} \tag{B.4}$$

### B.2.5 Proposition 2

Proposition 2 is to be proven:

$$\mathbb{E}[R_{[y]}(C_i) \wedge R_{[y]}(C_j)] = R_{[y]}(C_i \cup C_j) \iff R_{[y]}(C_i) = 0 \vee R_{[y]}(C_j) = 0$$

where  $R_{[y]}(C_j)$  are the  $B^3$  recall scores of label  $y$  on elements in a cluster  $C_j$ . Let  $|y|, C_i, C_j, |C_i|_y, |C_j|_y$  be defined as in Proof B.2.1, from which it holds:

$$\mathbb{E}[R_{[y]}(C_i) \wedge R_{[y]}(C_j)] \leq R_{[y]}(C_i \cup C_j) \iff |C_i|_y |C_j|_y \geq 0 \quad (\text{B.5})$$

and so, we can directly conclude that:

$$\begin{aligned} \mathbb{E}[R_{[y]}(C_i) \wedge R_{[y]}(C_j)] &= R_{[y]}(C_i \cup C_j) \\ \iff |C_i|_y |C_j|_y &= 0 \\ \iff |C_i|_y = 0 \vee |C_j|_y &= 0 \\ \iff R_{[y]}(C_i) = 0 \vee R_{[y]}(C_j) &= 0 \end{aligned} \quad (\text{B.6})$$

### B.2.6 Corollary 2

We prove Corollary 2 that:

$$\mathbb{E}[R_{[y]}(C_i) \wedge R_{[y]}(C_j)] < R_{[y]}(C_i \cup C_j) \iff R_{[y]}(C_i) \neq 0 \wedge R_{[y]}(C_j) \neq 0$$

Analogous to B.2.3, we know from Proposition 2 that if  $R_{[y]}(C_i) \neq 0 \wedge R_{[y]}(C_j) \neq 0$ , then  $\mathbb{E}[R_{[y]}(C_i) \wedge R_{[y]}(C_j)] \neq R_{[y]}(C_i \cup C_j)$ . Since Theorem 2 must hold, but equality is not given, then it must be that  $\mathbb{E}[R_{[y]}(C_i) \wedge R_{[y]}(C_j)] < R_{[y]}(C_i \cup C_j)$ .

### B.2.7 Generalization to Multiple Clusters

Let  $P_{[y]}(C_j), R_{[y]}(C_j)$  denote respectively the  $B^3$  precision and recall scores of label  $y$  on elements in a cluster  $C_j$ . It was shown earlier that Theorems 1 & 2 hold for any two pairwise clusters. Assume now (instead) having a fixed but arbitrary number of clusters  $C_1, C_2, \dots, C_k$ , with  $k > 2$ . The following grouping can now be iteratively applied for  $j > 1$ :

$$\begin{aligned} C_1^* &:= C_1 \\ C_j^* &:= C_j \cup C_{j-1}^* \end{aligned} \quad (\text{B.7})$$

Notice that  $C_j^* \triangleq \bigcup_{i < j} C_i$ . The above definition allows us to realize that, in fact, for any tuple pair  $(C_j, C_{j-1}^*)$  Theorems 1 & 2 must also hold. Thus, in particular, it holds for the pair  $(C_k, C_{k-1}^*)$ , and so:

$$\begin{aligned} \mathbb{E}[P_{[y]}(\bigcup_{j \leq k} C_j)] &\leq \mathbb{E}[P_{[y]}(C_{k-1}^*) \wedge P_{[y]}(C_k)] \\ &\leq \mathbb{E}[\mathbb{E}[P_{[y]}(C_{k-2}^*) \wedge P_{[y]}(C_{k-1})] \wedge P_{[y]}(C_k)] \\ &\leq \mathbb{E}[\mathbb{E}[\dots \mathbb{E}[P_{[y]}(C_1^*) \wedge P_{[y]}(C_2)] \dots] \wedge P_{[y]}(C_{k-1}) \wedge P_{[y]}(C_k)] \\ &\leq \mathbb{E}[P_{[y]}(C_1) \wedge \dots \wedge P_{[y]}(C_{k-1}) \wedge P_{[y]}(C_k)] \\ &\leq \mathbb{E}[\bigwedge_{j \leq k} P_{[y]}(C_j)] \end{aligned} \quad (\text{B.8})$$

and likewise, it holds that:

$$\mathbb{E}[R_{[y]}(\bigcup_{j \leq k} C_j)] \geq \mathbb{E}[R_{[y]}(C_{k-1}^*) \wedge R_{[y]}(C_k)] \geq \mathbb{E}\left[\bigwedge_{j \leq k} R_{[y]}(C_j)\right] \quad (\text{B.9})$$

### B.2.8 Proof of Coherence for $\alpha\text{Max-}B_\delta^3$

Let  $C_i, C_j$  be disjoint clusters of distinct elements. Theorem 1 is to be proven for the weighted version:  $\mathbb{E}[P_{[y]}^\delta(C_i) \wedge P_{[y]}^\delta(C_j)] \geq \mathbb{E}[P_{[y]}^\delta(C_i \cup C_j)]$ , where  $P_{[y]}^\delta(C)$  is the effective-number-of-samples (ENS) weighted  $B^3$  precision scores of label  $y$  on elements in  $C_j$  for a  $\delta_y \in [0, 1]$ ; with weight  $w_y := (1 - \delta_y^{|y|}) / (1 - \delta)$ .  $|y|$  be the frequency of the class  $y$  in the entire data set. Let  $|C_i|, |C_j| \in \mathbb{N}$  denote the total number of elements in  $C_i, C_j$  and  $|C_i|_y, |C_j|_y \in \mathbb{N}$  respectively the number of elements in clusters  $C_i, C_j$  with label  $y$ . Then, it holds:

$$\begin{aligned} \mathbb{E}[P_{[y]}^\delta(C_i) \wedge P_{[y]}^\delta(C_j)] &\geq P_{[y]}^\delta(C_i \cup C_j) \\ \iff \mathbb{E}[w_y^{-1} P_{[y]}(C_i) \wedge w_y^{-1} P_{[y]}(C_j)] &\geq w_y^{-1} P_{[y]}(C_i \cup C_j) \\ \iff w_y^{-1} \frac{|C_i|_y R_{[y]}(C_i) + |C_j|_y R_{[y]}(C_j)}{|C_i|_y + |C_j|_y} &\geq w_y^{-1} P_{[y]}(C_i \cup C_j) \\ \iff w_y^{-1} \mathbb{E}[P_{[y]}(C_i) \wedge P_{[y]}(C_j)] &\geq w_y^{-1} P_{[y]}(C_i \cup C_j) \\ \iff \mathbb{E}[P_{[y]}(C_i) \wedge P_{[y]}(C_j)] &\geq P_{[y]}(C_i \cup C_j) \\ \stackrel{\text{Proof of B.2.1}}{\iff} \top \end{aligned} \quad (\text{B.10})$$

Since the inverse weight term  $w_y^{-1}$  cancels out, the proof for Proposition 1 is also obtained immediately. Similarly, it can easily be shown that Theorem 2 for the weighted version  $\mathbb{E}[R_{[y]}^\delta(C_i) \wedge R_{[y]}^\delta(C_j)] \leq \mathbb{E}[R_{[y]}^\delta(C_i \cup C_j)]$  holds as well:

$$\begin{aligned} \mathbb{E}[R_{[y]}^\delta(C_i) \wedge R_{[y]}^\delta(C_j)] &\leq R_{[y]}^\delta(C_i \cup C_j) \\ \iff \mathbb{E}[w_y^{-1} R_{[y]}(C_i) + w_y^{-1} R_{[y]}(C_j)] &\leq w_y^{-1} R_{[y]}(C_i \cup C_j) \\ \iff w_y^{-1} \frac{|C_i|_y R_{[y]}(C_i) + |C_j|_y R_{[y]}(C_j)}{|C_i|_y + |C_j|_y} &\leq w_y^{-1} R_{[y]}(C_i \cup C_j) \\ \iff \mathbb{E}[R_{[y]}(C_i) \wedge R_{[y]}(C_j)] &\leq R_{[y]}(C_i \cup C_j) \\ \stackrel{\text{Proof of B.2.4}}{\iff} \top \end{aligned} \quad (\text{B.11})$$

Again, as  $w_y^{-1}$  cancels out, the respective proof for Proposition 2 follows directly.

### B.2.9 Proof of Non-Monotonicity

Following Equations 7.108, 7.110, let  $\mathbb{E}[\eta_\alpha^{[j]}]$  be a continuous function over the closed interval  $\alpha \in [0, 1]$  such that  $\lim_{\alpha \rightarrow 0} \mathbb{E}[\eta_\alpha^{[j]}] = 0^+$  and  $\lim_{\alpha \rightarrow 1} \mathbb{E}[\eta_\alpha^{[j]}] = \mathbb{E}[\eta^{[j]}]^+$ . We show that  $\mathbb{E}[\eta_\alpha^{[j]}]$  cannot be monotonic while approaching its limit from above at both endpoints. Suppose, for the sake of contradiction, that  $\mathbb{E}[\eta_\alpha^{[j]}]$  is a monotonic function. Since  $\lim_{\alpha \rightarrow 1} \mathbb{E}[\eta_\alpha^{[j]}] = \mathbb{E}[\eta^{[j]}]^+$ , there exists an  $\alpha' \in [0, 1]$  such that  $0^+ \leq \mathbb{E}[\eta_{\alpha'}^{[j]}] \leq \mathbb{E}[\eta^{[j]}]^+$ . By the Intermediate Value Theorem, there must exist an  $\alpha'' \in [0, 1]$

such that  $\mathbb{E}[\eta_{\alpha''}^{[j]}] = \mathbb{E}[\eta^{[j]}]$ . However, this contradicts the assumption that  $\mathbb{E}[\eta_{\alpha}^{[j]}]$  approaches its limit from (only) *above* at both endpoints. Therefore, it cannot be monotonic.

### B.2.10 Theorems' Inequality Implications

Let  $\mathbf{S}_i$  be a set of clusters  $C_j \subset S_i$ , where  $S_i$  is the union of all the elements in the clusters; such that  $S_i := \bigcup_{C_j \in \mathbf{S}_i} C_j$ . Hence,  $\mathbf{S}_i$  is a set of clusters; whereas  $S_i$  is a set of elements. We show that Theorems 1, 2 imply that  $\mathbb{E}_{C_j \in \mathbf{S}_i} \left[ \frac{P_{[y]}(C_j)}{P_{[y]}(S_i)} \right] \geq 1$ ; and  $\frac{R_{[y]}(C_j)}{R_{[y]}(S_i)} \leq 1$ . Let  $|\mathbf{S}_i|, |S_i|, |C_j| \in \mathbb{N}$  denote the number of elements and  $|S_i|_y, |C_j|_y \in \mathbb{N}$  respectively the number of elements with label  $y$ .  $|S_i| \geq |C_j|$  and  $|S_i|_y \geq |C_j|_y$  is guaranteed since  $C_j \subset S_i$ .  $|y| \in \mathbb{N}$  is the total number of elements with label  $y$  in the entire data set.

Starting with *recall*, where:

$$\frac{R_{[y]}(C_j)}{R_{[y]}(S_i)} = \frac{|C_j|_y}{|y|} / \frac{|S_i|_y}{|y|} = \frac{|C_j|_y}{|S_i|_y} \leq 1. \quad (\text{B.12})$$

holds trivially per definition since  $|S_i|_y \geq |C_j|_y$ .

For *precision*, we know:

$$\mathbb{E}_{C_j \in \mathbf{S}_i} \left[ \frac{P_{[y]}(C_j)}{P_{[y]}(S_i)} \right] \triangleq \frac{1}{|\mathbf{S}_i|} \sum_{C_j \in \mathbf{S}_i} \frac{P_{[y]}(C_j)}{P_{[y]}(S_i)} \quad (\text{B.13})$$

and therefore,

$$\begin{aligned} & \mathbb{E}_{C_j \in \mathbf{S}_i} \left[ \frac{P_{[y]}(C_j)}{P_{[y]}(S_i)} \right] \geq 1 \\ \iff & \frac{1}{|\mathbf{S}_i|} \sum_{C_j \in \mathbf{S}_i} \frac{P_{[y]}(C_j)}{P_{[y]}(S_i)} \geq 1 \\ \iff & \frac{1}{|\mathbf{S}_i|} \sum_{C_j \in \mathbf{S}_i} P_{[y]}(C_j) \geq P_{[y]}(S_i) \\ \iff & \frac{1}{|\mathbf{S}_i|} \sum_{C_j \in \mathbf{S}_i} \frac{|C_j|_y}{|C_j|} \geq \frac{|S_i|_y}{|S_i|} \end{aligned} \quad (\text{B.14})$$

We can re-write the expression  $\frac{|S_i|_y}{|S_i|}$  as:

$$\frac{|S_i|_y}{|S_i|} = \frac{\sum_{C_j \in \mathbf{S}_i} |C_j|_y}{\sum_{C_j \in \mathbf{S}_i} |C_j|} = \sum_{C_j \in \mathbf{S}_i} \frac{|C_j|_y}{\sum_{C_k \in \mathbf{S}_i} |C_k|} \quad (\text{B.15})$$

and since  $\sum_{C_k \in \mathbf{S}_i} |C_k| \geq \mathbb{E}_{C_j \in \mathbf{S}_i} [|C_j|] = \frac{1}{|\mathbf{S}_i|} \sum_{C_k \in \mathbf{S}_i} |C_k|$ , we obtain the inequality:

$$\sum_{C_j \in \mathbf{S}_i} \frac{|C_j|_y}{\sum_{C_k \in \mathbf{S}_i} |C_k|} \leq \sum_{C_j \in \mathbf{S}_i} \frac{|C_j|_y}{\frac{1}{|\mathbf{S}_i|} \sum_{C_k \in \mathbf{S}_i} |C_k|} \quad (\text{B.16})$$

and so, with  $\frac{1}{|\mathbf{S}_i|} \sum_{C_k \in \mathbf{S}_i} |C_k| \geq \frac{1}{|\mathbf{S}_i|} |C_j|$  for all  $j$ , we have:

$$\frac{1}{|\mathbf{S}_i|} \sum_{C_j \in \mathbf{S}_i} \frac{|C_j|_y}{|C_j|} \geq \sum_{C_j \in \mathbf{S}_i} \frac{|C_j|_y}{\frac{1}{|\mathbf{S}_i|} \sum_{C_k \in \mathbf{S}_i} |C_k|} \geq \frac{|\mathbf{S}_i|_y}{|\mathbf{S}_i|} \quad (\text{B.17})$$

which proofs  $\mathbb{E}_{C_j \in \mathbf{S}_i} \left[ \frac{P_{[y]}(C_j)}{P_{[y]}(\mathbf{S}_i)} \right] \geq 1$ .

### B.2.11 Proof of $\alpha$ -Plateau

We prove the existence of a function plateau in Equation 7.110 by proving that:

$$f(\alpha) := \frac{\min[p, \alpha q] \min[p_2, \alpha q_2]}{\max[p, \alpha q] \max[p_2, \alpha q_2]} \quad (\text{B.18})$$

has a plateau  $f(\alpha_1) = f(\alpha_2)$  in  $\min[a, a_2] \leq \alpha_1 \leq \alpha_2 \leq \max[a, a_2]$  for  $\alpha_1, \alpha_2 \in [0, 1]$ ; given  $\nu_1 = \frac{p_1}{q_1}$ ,  $\nu_2 = \frac{p_2}{q_2}$ , and fixed  $p_1, q_1, p_2, q_2$ . First, we consider the fact that:

$$g_{p,q}(\alpha) := \frac{\min[p, \alpha q]}{\max[p, \alpha q]} = \begin{cases} \frac{\alpha q}{p} & \text{if } \alpha \leq \frac{p}{q} \\ \frac{p}{\alpha q} = \left(\frac{\alpha q}{p}\right)^{-1} & \text{otherwise} \end{cases} \quad (\text{B.19})$$

which holds since  $\min[p, \alpha q] = \alpha q \iff \alpha q \leq p \iff \alpha \leq \frac{p}{q}$ , and also since  $\max[p, \alpha q] = -\min[-p, -\alpha q]$ . W.l.o.g, assume  $\nu_1 \leq \nu_2$ , and let  $\alpha \in [\nu_1, \nu_2]$ . Since  $\alpha_1 \geq \nu_1$ , it holds  $g_{p_1, q_1}(\alpha) = \left(\frac{\alpha q_1}{p_1}\right)^{-1}$ . Similarly,  $\alpha \leq \nu_2 \implies g_{p_2, q_2}(\alpha) = \frac{\alpha q_2}{p_2}$ . Therefore,  $\nu_1, \nu_2$  are the two inversion points, and so:

$$f(\alpha) = g_{p_1, q_1}(\alpha) \cdot g_{p_2, q_2}(\alpha) = \frac{p_1}{\alpha q_1} \frac{\alpha q_2}{p_2} = \frac{p_1 q_2}{q_1 p_2} = \nu_1 \nu_2 \quad (\text{B.20})$$

is independent of  $\alpha$  within that interval and, thus, *constant* (i.e. a plateau).



## Appendix C

# From Missteps to Milestones: Understanding Failed Attempts

In this appendix section, we have included some initial prototypes and algorithm concepts that were proposed in the early stages of development. We will discuss some failed attempts and prototypes that were ultimately unsuccessful, but we still acknowledge their importance as they served as crucial first-step approaches and lessons learned. Despite failing to achieve the desired outcome/effectiveness, these prototypes played a crucial role in exploring boundaries, providing interesting insights, and shaping the way of our subsequent ideas and solutions.

### C.1 Failed Prototype Attempt: Automated Textual Description Generation

The following prototype idea was outperformed by the method presented in Chapter 7.10 and came with some issues. In the following, we explain the prototype and the reason for its failure and weaknesses.

#### C.1.1 Methods

The initial algorithm was similar to the architecture of 7.10 but with some differences in the second phase and training objective. In the second phase, i.e. the cluster captioning, the vision encoder and text decoder modules of ViED were here frozen, while the language modeling head (LMHead) was trained. The image clusters were identified using similarity or distance metrics on input images or clustering algorithms on the hidden states of the vision encoder’s output. Some naive clustering approaches here were k-means, hierarchical methods, and density-based clustering.

The CLIP score similarity matrix was calculated for the hidden vision states and the hidden states of the CLIP text encoder. The average CLIP score similarity to all captions was determined for each instance’s hidden state. The scores produced a weighted averaged projected pooling representation; then received by the decoder and passed to the ViED’s LMHead. The LMHead predicted the next word in a sequence of words based on the decoder’s outputs and was fine-tuned by optimizing the prediction of all instance captions (of the cluster; via a single cluster representation). By fine-tuning the LMHead we intended to align the pooled embedding to be manifold-compatible and optimize the training loss (i.e. the similarity).

The CLIP weighting method was used to produce a description that considered the most representative embeddings more strongly. This means that the loss predictions of individual instance captions were weighted by the CLIP scores of the respective hidden states of the text and vision encoders. Essentially, the goal was to

train a model to generate textual descriptions that were most similar to those images and captions that were most similar to all other instances in the cluster. This way, so we assumed, the model would be able to provide more accurate and relevant descriptions (yet, this was *not* effective enough; as turned out later).

Figure C.1 shows the architectural overview of this (failed) prototype approach.

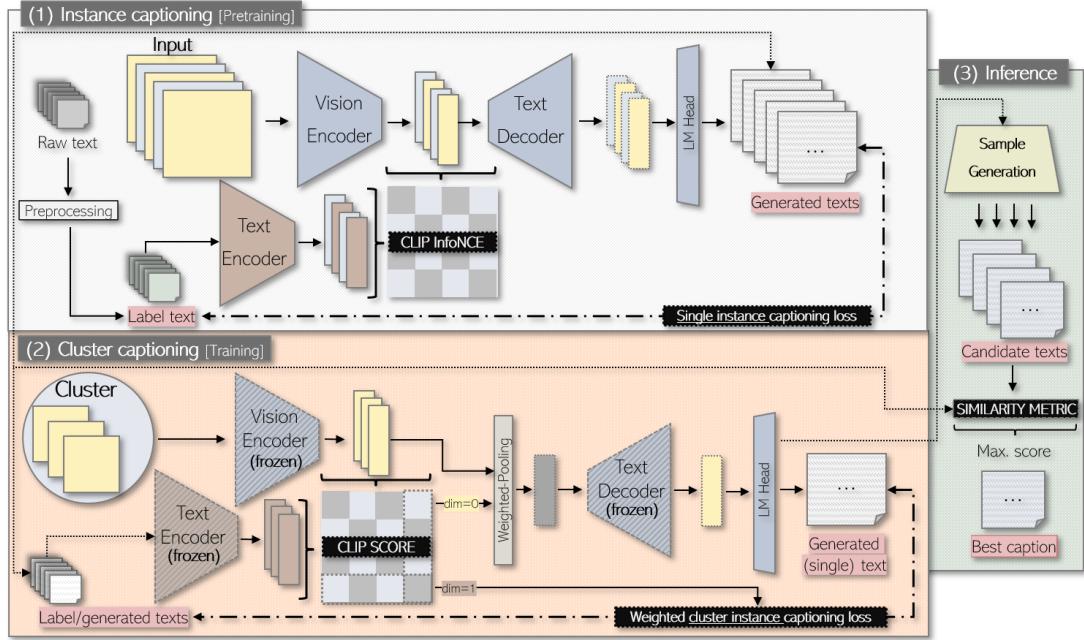


FIGURE C.1: An overview of an initial prototype training architecture pipeline (i.e. first attempt) for generating cluster descriptions.

The CLIP score weighting matrix  $S$ , for a given cluster  $C_m$  (with  $n$  elements) and latent vision states  $Z_m$  and text states  $Z_m^{text}$ , was  $S := \text{CLIP-SCORE}(Z_m^{text}, Z_m)$ ; with shape  $n \times n$ . The image-to-texts and text-to-images CLIP scores were respectively  $s_0 := \sum_{i=1}^n S^{[i,:]}$  and  $s_1 := \sum_{i=1}^n S^{[:,i]}$ . The corresponding normalized probability scores were  $\hat{s}_0, \hat{s}_1$ . The vector pooling was the weighted feature average of  $Z_m$  by  $s_0$ , given by:  $\sum_{i=1}^n \hat{s}_0^{[i]} Z_m^{[:,i]}$ .

This CLIP-score-weighted vector pooling operation is denoted by  $\bullet[.]$ , so that  $\bullet[Z_m] := v_m^\bullet$ . Since encoders  $E, E_{text}$  were frozen during training,  $v_m^\bullet$  was constant and could be cached. Similarly, for a given text  $t_i$ , the LMHead prediction loss  $l_{|\varrho}^{[i]}$  was weighted by  $\hat{s}_1^{[i]}$ , where  $i = 1, \dots, n$  and  $\varrho$  is the LMHead text decoding strategy. So, given  $C_m$ , instances within the cluster were first processed by encoder  $E$ , resulting in a set of latent vectors  $Z_m$ . These vectors were then averaged using a CLIP score weighted pooling operation to obtain a new latent cluster representation  $v_m^\bullet$ , which was then forwarded through a decoder  $D_{|\varrho}$  with decoding strategy  $\varrho$  to:

$$\hat{y}_m = D_{|\varrho}(v_m^\bullet) \triangleq D_{|\varrho}(\bullet[E(C_m)]) \quad (\text{C.1})$$

where only the LMHead (of decoder  $D$ ) is trainable.

### Training objective

Let  $\varrho^\nabla, \varrho^\Delta$  respectively denote the text decoding strategy during training ( $\nabla$ ) and inference ( $\Delta$ ). The objective was to approximate the prediction of all instance captions

within a given cluster  $C_m$ , weighted by their importance. Ideally, for all instances within  $C_m$ , we want:  $\forall_{x_i \in C_m} : t_i \equiv D_{|\varrho^\triangle}(v_m^\bullet)$ .

The decoding strategy  $\varrho$  behaves differently during training and inference. Using only the latent cluster representation  $v_m^\bullet$  on decoder  $D_{|\varrho^\triangle}$  to predict all instance captions within cluster  $C_m$  is expected to be less accurate than using decoding strategy  $\varrho^\nabla$ . This is because ground-truth context is not available during inference and a single fixed representation may not capture the diversity within a cluster. Instances within a cluster are similar but *not* identical and can have very diverse captions.

So in summary, to obtain a good overall representation during inference (i.e. for decoding strategy  $\varrho^\triangle$ ) we optimized training using the decoding strategy  $\varrho^\nabla$ . We minimized the difference between the predicted cluster description and all instance captions within a given cluster; creating a representation that aligns with the cluster's instance captions and is compatible with the decoder's manifold.

### Cluster-wise gradient accumulation

Small batch sizes often have noisier gradients due to stochastic data sampling, but they can result in better parameter space exploration. Greater batch sizes can yield more accurate gradient estimates, but they can cause overfitting and poor generalization. To maintain training stability in our specific scenario of creating cluster-wise descriptions where generalization is not important (we want an optimal solution for a specific cluster), we favored accuracy over generalization; and hence chose bigger batch sizes, ideally the entire cluster. Yet, very large cluster sizes might cause performance concerns and out-of-memory errors. Thus, we used cluster-wise gradient accumulation, which involved randomly splitting a cluster into smaller subgroups with a predetermined batch size, computing gradient accumulation for each subgroup, and finally updating the model with the accumulated gradients.

### Prototype discussion

The reason why the prototype failed to perform as desired can be attributed to two main factors, namely, manifold destruction and label overfitting. The initial idea was to freeze the encoder and decoder modules so that the learned manifolds were maintained and not altered. However, by fine-tuning the LMHead, the manifold could degenerate again, resulting in decreased performance. This happened because the pre-trained model with the LMHead was already well-aligned and fitted on a large corpus of pre-training data, having highly encoded information, semantic meanings, and language generation logic. Fine-tuning the model on a specific cluster of instances, by overfitting, resulted in the destruction of that strong manifold. Consequently, even though all the modules were frozen, the last linear mapping destroyed the semantic manifold (i.e. failed to interpret it correctly); leading to poor results.

The second factor that contributed to the failure of the prototype was label overfitting (i.e. the training loss). Training a model to predict all the labels of all the cluster instances was expected to learn a representation that would more or less fit all of them on average. However, this was not the case due to the way training and inference work in LLMs: predictions are context-based, starting with the most likely word, followed by the next most likely word given a previous word, sequence, context, etc. During training, this previous context is the previous context in the ground-truth (label) sequence; however, during inference, it is the previous context predicted by the model itself. Thus, the LMHead learned precisely this, to predict captions given the right ground-truth *context* (as this was the loss function it

was optimized for). This is a problem since the model was optimized (i.e. overfitted) to produce a description that fits specific images given their ground-truth context, which shaped the manifold accordingly; but this did not extrapolate to our intended use-case. In other words, this means the model was not generating general captions but was overfitted to reproduce specific captions whenever given the label context. Therefore, the model had the wrong training objective loss.

## C.2 Sensor-Driven Textual Observation Report Prediction

When we have complex sensor data and corresponding textual reports, we may wonder if we can create a model that can establish a relationship between the two domains. This would enable us to automatically gather sensor data and produce a corresponding textual report (which is what we expected; but was overall not effective and reliant enough). To achieve this, we tried to train a (correlation-based) decoder model. The training idea was to essentially predict the word sequences in the textual report using only the sensor data as model input. Achieving this would allow us to generate corresponding (i.e. matching) free-form textual reports directly out of the sensor data. An illustration is given in Figure C.2.

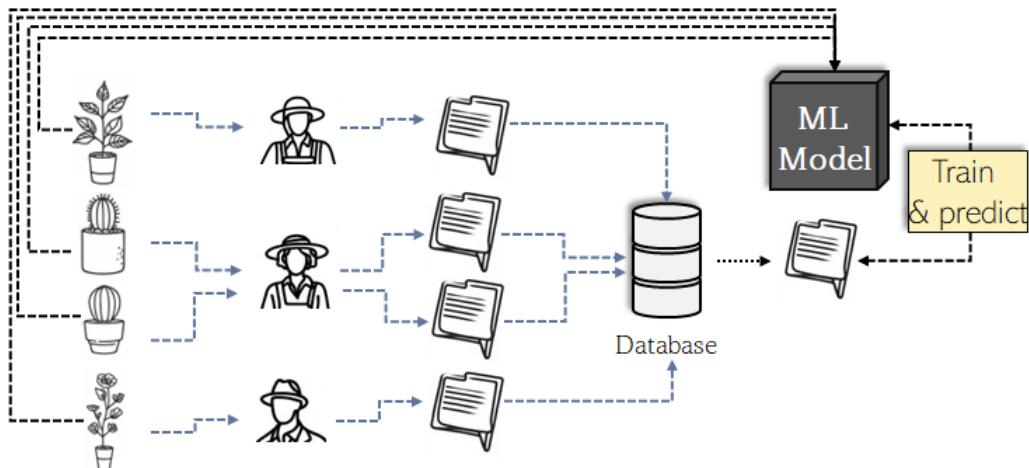


FIGURE C.2: Learning from textual observations and training decoder models for automatic text generation out of sensor data.

### C.2.1 Prototype Discussion

A main and common problem of LLMs is that they can (sometimes, often, etc.) generate text that is *not* accurate or factual, but appears to be plausible and correct. This is known as "hallucination"; and dangerous. Hallucination occurs because conventional LLMs have a limited/conditioned understanding and are unable to verify the truth of their output. In general, it is difficult to regulate hallucination (precisely) due to a model's limited contextual understanding. Hallucination can be confusing, harmful, and misleading, so LLMs should not be used in certain applications unless the model quality is stable and reliable.

In our experiments, we were able to generate textual reports that domain experts found indistinguishable from real ones at first glance. Yet, extensive fact-checking and analysis were necessary to determine their correctness; since LLM hallucination could (and did) also occur. However, due to the unreliability, this approach was

deemed unsuitable and not of interest anymore (in such a current uncertain state of potential hallucinations). The idea of producing keywords instead of free text also does not fully solve or avoid this problem of uncertainty.

To reduce the severity of hallucinations in a naive manner, we suggest the following steps: cleaning the data, fine-tuning it with control, improving contextual understanding, implementing basic verification methods, and ideally providing transparent explanations. Some simple verification techniques could include: (1) *Fact-checking*: Quickly searching the database to verify the accuracy of the generated output, which can help to identify (obvious) errors and misleading information. (2) *Keyword matching*: Checking if certain expected keywords or phrases are present in the generated output, e.g. ensuring so relevance to the topic at hand. (3) *Avoiding or filtering out overly specific information*: It is safer to provide more general information rather than overly specific details. For example, stating that a water plant was dehydrated is preferable to specifying the exact duration of dehydration. (4) *Logical consistency checks*: Assessing the coherence and consistency of the generated output, ensuring that it makes logical sense and does not contradict itself.

Still, it is crucial to acknowledge that these verification methods do *not* completely solve the issue of hallucination but rather provide some basic safety checks. They still come with many limitations and will certainly not detect all inaccuracies. It would be absolutely wrong and naive to assume otherwise. More advanced mechanisms are definitely necessary to ensure reliability and precision.



## Appendix D

# Deutsche Zusammenfassung

**Note:** According to the examination regulations, it is necessary for non-German documents to include a German summary. Therefore, a *translated* summary of this thesis is provided at this point.

**Hinweis (1):** Gemäß den Bestimmungen der Prüfungsordnung ist es notwendig, dass Dokumente, die nicht auf Deutsch verfasst sind, eine Zusammenfassung in deutscher Sprache enthalten. Daher wird an dieser Stelle eine *übersetzte* Zusammenfassung der Arbeit präsentiert.

**Hinweis (2):** Diese Arbeit wurde durch eine gemeinsame Zusammenarbeit der BMW Group und der Ludwig-Maximilians-Universität München im Rahmen einer industriellen Doktorarbeit ermöglicht. Alle (eventuell) möglichen datenschutzbezogenen Informationen wurden entweder ausgelassen oder gemäß den geltenden Richtlinien der BMW Group anonymisiert.

### D.1 Zusammenfassung

Die steigende Nachfrage nach der Integration von Künstlicher Intelligenz (KI) erfordert effiziente und zuverlässige Lösungen. Innerhalb des KI-Feldes ist das bekannte Maschinelle Lernen (ML) ein sehr dynamischer Teilbereich. ML basierte Methoden und Lösungsansätze entwickeln sich ständig und rasant weiter, um immer leistungsfähigere und skalierbarere Algorithmen für zunehmend komplexe Probleme und Anforderungen zu schaffen. Diese Arbeit konzentriert sich insbesondere auf Herausforderungen des ML im Bereich automatisierte Entscheidungsfindung und Mustererkennung. Dabei werden verschiedene Methoden für das effektive Erlernen von Datenrepräsentationen vorgestellt und analysiert.

Die wissenschaftlichen Beiträge dieser Arbeit umfassen eine Vielzahl von Algorithmen und Verbesserungen, sowie etwa Ausreißererkennungsalgorithmen, Clusterbildungsmethoden oder graphenbasierte ML-Modelle. Das übergeordnete Ziel dieser Arbeit ist es, KI-Methoden, insbesondere für relevante industrielle Anwendungen, anzupassen, verbessern, oder weiterzuentwickeln.

#### D.1.1 Forschungsmotivation

Diese Arbeit entstand aus einem früheren Datenanalyseprojekt, das das Potenzial von KI-Methoden in Bezug auf Effizienz und Effektivität aufzeigte. Dies führte und motivierte zur aktuellen Arbeit, die zunächst darauf abzielte, das vorherige Modell und die Methodik zu verbessern oder vollständig neu zu gestalten, um noch robustere und effizientere Ergebnisse zu erzielen. Starke und skalierbare KI-Lösungen

sind aufgrund ihrer breiten Anwendbarkeit und Anpassungsfähigkeit äußerst wünschenswert und in der Industrie sehr gefragt. Beispiele hierfür sind die Automatisierung von Vorhersagen, Klassifizierung, semantische Ähnlichkeitssuche, Ausreißererkennung und vieles mehr. In diesem Zusammenhang spielen Datenrepräsentationen eine entscheidende Rolle. Diese stellen kontextualisierte numerische Darstellungen von Entitäten, bzw. Objekten dar. Die Generierung guter numerischer Datenrepräsentationen ist herausfordernd, ermöglicht aber Anwendungen in vielen verschiedenen Domänen und ist daher sehr wertvoll.

### D.1.2 Forschungsbereich

Der in dieser Arbeit behandelte Forschungsbereich umfasst mehrere Bereiche der KI: sowie ML, Deep Learning (DL), Representation Learning, Manifold Learning, Ausreißererkennung, Clustering, etc. Der Schwerpunkt jedoch liegt klar auf dem "Erlernen von Datenrepräsentationen", was bedeutet, dass starke und effiziente Darstellungen für komplexe, unkonventionelle Datenarten und -strukturen gefunden werden sollen, um bestimmte Aufgaben (optimal) zu lösen. Aus dieser Hauptrichtung haben sich dann verschiedene Nebenrichtungen entwickelt, die sich mit verschiedenen Unterthemen und Nachfolgeaufgaben befassen, aber stets entweder direkt oder indirekt mit Representation Learning oder Manifold Learning verbunden sind; somit immer im Bereich KI (bzw. ML).

### D.1.3 Forschungsherausforderungen

Evaluierung ist ein grundlegender Aspekt wissenschaftlicher Forschung. Nur so ist es möglich, die Qualität und Effektivität von Methoden qualitativ zu bewerten und sie mit anderen Methoden zu vergleichen. Die Evaluierung von KI Methoden ist jedoch herausfordernd aufgrund unvermeidbarer Kompromisse; jede Methode hat verschiedene Vor- und Nachteile. Unterschiedliche Methoden zeichnen sich typischerweise in verschiedenen Aufgaben und Bereichen aus, wobei einige effizienter, aber weniger effektiv sind, und umgekehrt. Daher ist eine "gerechte" Evaluierung nicht immer einfach und offensichtlich. Falsche Bewertungen können leicht zu falschen Schlussfolgerungen führen. Bspw. zeigen in der Clusteranalyse verschiedene Algorithmen unterschiedliche Lösungen auf, oftmals hängt hier die Effektivität direkt an der Verteilung der Daten ab. Es gibt keinen universellen Algorithmus, der alle anderen Methoden in allen möglichen Aspekten übertrifft. Um zuverlässige und klare Schlussfolgerungen zu ziehen, sind in der Regel eine Vielzahl von Experimenten und Wiederholungen erforderlich. Darüber hinaus ist es entscheidend, dass empirische Ergebnisse und Experimente (wenn möglich) durch mathematische Theoreme oder Beweise gestützt werden. Insbesondere bei der Verwendung von "Beweisen" sind empirische Ergebnisse möglicherweise nicht mehr erforderlich, da sie bestimmte Eigenschaften garantieren. Im Allgemeinen gibt es somit zwei Ansätze, um sicherzustellen, dass bestimmte Eigenschaften signifikant sind: Entweder führt man ausreichend viele Experimente durch und sammelt empirische Ergebnisse um Schlussfolgerungen statistisch zu stützen, oder man verwendet direkte mathematische Beweise und Ableitungen, um bestimmte Eigenschaften mathematisch zu garantieren. In dieser Arbeit wurde daher ein angemessenes Gleichgewicht zwischen mathematischen Beweisen und empirischen Ergebnissen berücksichtigt.

#### D.1.4 Beitragsumfang der Arbeit

Die in dieser Arbeit vorgestellte Forschung stellt einen wesentlichen Teil eines industriellen Doktorandenprogramms dar, das Theorie mit praktischen Anwendungen kombiniert. Die Arbeit konzentriert sich speziell darauf, die theoretische und analytische Grundlage innerhalb dieses Projektumfelds zusammenzufassen und enthält daher keinerlei datenschutzempfindliche Informationen. Sie präsentiert hauptsächlich theoretische Lösungen und Ansätze die entwickelt wurden, um das zugrunde liegende Forschungsproblem anzugehen. Alle Resultate, die in dieser Arbeit präsentiert werden, stammen ausschließlich aus öffentlich allen zugänglichen (z.b. akademischen) Datensätzen. Der Fokus dieser Arbeit liegt somit ausschließlich darauf, relevantes Forschungswissen bereitzustellen, weshalb insbesondere keine Implementierungs- oder Anwendungsdetails des Projektes enthalten sind; in Übereinstimmung mit der imponierten Vertraulichkeitsvereinbarung.

Die Hauptbeiträge dieser Arbeit werden in den Methoden- und Ergebniskapiteln sowie in den jeweiligen Diskussionen dargestellt und erläutert. Die Kapitel Motivation, Einleitung und verwandte Arbeiten dienen dazu, einen umfassenden Überblick über zuvor veröffentlichte Arbeiten, Forschungen, etc. aus der entsprechenden wissenschaftlichen Literatur zu geben. Dies soll gemeinsames Hintergrundwissen vermitteln, welches für die Einordnung dieser Arbeit in den Kontext als grundlegend erachtet wird. Für alle verwandten Arbeiten und einführenden Kapitel werden umfangreiche Referenzen angegeben.

#### D.1.5 Zusammenfassung der Beiträge

Zusätzlich zum bereits erwähnten Beitrag eines umfangreichen Literaturüberblicks sind im Folgenden insbesondere die originalen *neuartigen* Beiträge der einzelnen Hauptmethoden dieser Arbeit explizit zusammengefasst. Diese gliedern die Arbeit in verschiedene Hauptbereiche:

### NEURONALE ARCHITEKTUR UND DATENREPRÄSENTATION

- **Einführung einer neuen nichtlinearen neuronalen Architektur basierend auf komplexen Zahlen (siehe 7.1):** Herkömmliche neuronale Netze verwenden reellwertige Parameter zur Verarbeitung und Analyse von Eingabedaten. Anstelle dessen wird hier nun eine komplexwertige neuronale Architektur präsentiert, bei der die neuronalen Parameter auf komplexen Zahlen basieren. Die Architektur weist im Vergleich zu traditionellen neuronalen Netzwerken mehrere einzigartige und interessante Eigenschaften auf; insbesondere die Fähigkeit, dass neuronale Eingaben niemals durch Gewichte überschrieben oder eliminiert werden können.
- **Entwicklung eines algorithmischen Strukturkonzepts zur Codierung hierarchischer 1:n Datenbanken in semantisch kontextualisierte numerische Datenrepräsentationen (siehe 7.2):** 1:n Datenbanken sind in vielen Bereichen gängig, stellen jedoch oft eine Herausforderung dar wenn es darum geht sie effektiv zu codieren und für ML vorzubereiten. Unser Ansatz verwendet eine hierarchische Datenrepräsentation, die eins-zu-viele-Beziehungen (1:n Datenbanken) in eine vereinheitlichte Baumstruktur überführt, die für ML kompatibel ist und eine Anwendung von KI-Methoden darauf effizient ermöglicht. Dies erlaubt uns somit insbesondere 1:n Datenbanken effizient semantisch und kontextualisiert zu kodieren, verarbeiten, und zu analysieren.

- **Vorstellung einer adaptiven und robusten Normalisierungs- und Vorverarbeitungsmethode (siehe 7.3):** Normalisierungs- und Vorverarbeitungstechniken sind entscheidend für die Verbesserung der Leistung von ML-Algorithmen. Der Fokus unserer neuen Methode liegt darauf, die Robustheit und Konvergenzfähigkeit des Trainings zu verbessern, indem der Einfluss von Ausreißern reduziert und die Feature-Skalierung verbessert wird.
- **Vorstellung einer Methode zur synthetischen Datengenerierung auf hierarchischen Baumstrukturen (siehe 7.4):** Datenanreicherung in der KI ist wichtiges Konzept zur Verbesserung der Leistung von maschinellen Lernalgorithmen. Unsere Methode generiert synthetische Datenbeispiele für hierarchische Baumstrukturen für 1:n Datenbanken. Dieser Ansatz ermöglicht es uns, die Anzahl und Vielfalt der Trainingsdaten zu erhöhen, mit dem Ziel die Leistung der auf diesen Daten trainierten Modelle zu verbessern.

## DIMENSIONSREDUKTION UND DARSTELLUNGSSUCHE

- **Vorstellung eines kontrastiven Frameworks zur Erlernung kontextueller Repräsentationen für hierarchische Baumstrukturen (siehe 7.5):** Eine sinnvolle, gute numerische Darstellung von Daten ist entscheidend für den Erfolg vieler KI-Modelle. Unser kontrastiver Ansatz zielt speziell darauf ab, repräsentativere Datenstrukturen zu erlernen; insbesondere für hierarchische Baumstrukturen mit 1:n Beziehungen. Der Ansatz ist äußerst flexibel anpassbar und kann sowohl unüberwacht, halbüberwacht, überwacht als auch selbstüberwacht trainiert werden. Die Methode zeigt Potenzial, bedeutungsvolle Datenrepräsentationen für komplexe hierarchische Datenbankentitäten zu extrahieren. Dadurch kann die Qualität weiterer nachgelagerter Aufgaben verbessert werden; wie Klassifikation, Ausreißererkennung, Clustering usw. Es ermöglicht insbesondere, kontextualisierte und bedeutsame numerische Repräsentationen automatisch und effizient zu erlernen.
- **Ausweitung und Verallgemeinerung des hierarchischen Datenbank Frameworks zur Vereinheitlichung mehrerer Datenansichten durch das Erlernen konkatenierter Darstellungen. (siehe 7.6):** Das Konkatenieren von Repräsentationen ist ein klassischer Lösungsansatz im Repräsentationslernen, um Datenrepräsentationen aus verschiedenen Datenquellen zusammenzuführen, wodurch eine einheitliche Gesamtrepräsentation erstellt wird. Von dieser Gesamtrepräsentation aus wird dann eine finale kompaktere Repräsentation erlernt oder abgeleitet. Anliegend daran besteht unsere Lösung darin, mehrere Datenansichten aus verschiedenen hierarchischen Datenbanken zu einer einzigen vereinheitlichten Datenbank zusammenzuführen. Dies geschieht zunächst durch das individuelle Erlernen und Berechnen der einzelnen Datenbankrepräsentationen für jede gegebene hierarchische Datenansicht. Anschließend wird aber eine gemeinsame Finalrepräsentation erlernt (die beispielsweise für nachgelagerte Aufgaben verwendet werden kann), wo aber dabei der Gradient nicht über alle Baumknoten bei der Backpropagation berechnet und verwendet wird, sondern stattdessen nur oberhalb eines Gradienten-Schnitts am obersten Wurzelknoten durchgeführt wird. Durch diese Vorgehensweise besteht Skalierbarkeit und Anpassungsfähigkeit des Frameworks auch für größere und komplexere Datenbanken, da das Erlernen der finalen Gesamtrepräsentation effizient ist.

- **Entwicklung von Methoden und diverser Zielfunktionen für das Erlernen von Datenbankrepräsentationen und Clustering (siehe 7.7):** Das Erlernen von Datenrepräsentationen und das Clustering dieser Repräsentationen sind von großer Bedeutung für hochdimensionale Daten mit komplexen Strukturen. Unsere Methoden verwenden Manifold-Learning-Techniken um semantische Repräsentationen zu erlernen, die die Dimensionalität kompakt reduzieren, aber gleichzeitig wichtige Informationen beibehalten und kodieren. Darüber hinaus wenden wir Clustering-Algorithmen auf den erlernten Datenbankrepräsentationen an, um große Datensätze in verschiedene zusammenhängende Gruppen zu unterteilen. Durch die Anwendung mehrerer Zielfunktionen in einer speziell für 1:n Datenbanken konzipierten neuronalen Trainingsarchitektur, erreichen wir effektive semantische, speziell optimierte, Datenrepräsentationen.
- **Entwicklung und Vorstellung von Ansätzen zur automatisierten Generierung textueller Beschreibung von Clustergruppen (siehe 7.10):** Clustergruppen sind generell herausfordernd zu interpretieren, insbesondere wenn es sich um hochdimensionale Daten handelt. Wir entwickeln und präsentieren daher Methoden zur automatisierten und generativen Erzeugung textueller Beschreibungen. Diese können wertvolle Einblicke in die Charakteristika verschiedener Clustergruppen liefern und menschlichen Nutzern dabei helfen, die gemeinsamen Eigenschaften der Cluster besser zu verstehen.

## DATENINTEGRATION UND ANALYSE

- **Formulierung und Einführung einer Evaluationsmetrik für Clustering unter Berücksichtigung von Label-Unsicherheit (siehe 7.9.2):** Clustering unter Label-Unsicherheit stellt ein weit verbreitetes und anspruchsvolles Problem dar, das in zahlreichen realen Anwendungen und Datensätzen häufig auftritt. Wir präsentieren daher eine modifizierte Version einer etablierten Evaluationsmetrik des überwachten Clusterings, und fokussieren speziell darauf, Clusteringresultate unter Berücksichtigung von Labelunsicherheit zu bewerten; insbesondere wenn die Labels vor allem größere Obergruppen andeuten und daher nicht exakt sind.
- **Vorstellung einer Metrikheuristik zur Bestimmung repräsentativer textueller Labels; für präzise Sensordaten mit ungenauen Labels (siehe 7.12):** Präzise Sensordaten mit ungenauen (textuellen) Labels stellen eine Herausforderung dar, die auch hier wieder in vielen realen Anwendungen auftritt. Unsere vorgeschlagene Metrikheuristik zielt darauf ab, aussagekräftige textuelle Labels (z.B. Kommentare) zu identifizieren, welche die Merkmale präziser und genauer Sensordaten repräsentieren. Dies gilt insbesondere auch dann, wenn der Datensatz ungenaue Labels aufweist und somit die Qualität der textuellen Daten (bzw. Labels) stark variiert.
- **Präsentation eines Transferlernansatzes zur Extraktion von Erkenntnissen aus verschiedenen ML-Modellen, die auf unterschiedlichen Datensätzen trainiert wurden (siehe 7.13):** Transferlernen ist eine bekannte und effektive Strategie, um das erlangte "Wissen" aus einem bestimmten Datensatz zu nutzen und es auf andere Datensätze zu übertragen um die Leistung von ML-Modellen zu steigern, die auf diesen anderen Datensätzen trainiert werden. Basierend

darauf stellen wir einen automatisierten Ansatz vor, um dies speziell für 1:n Datenbanken durchzuführen und dabei (semi-)überwacht zu trainieren.

- **Einführung einer neuen Methode zur Erkennung von Abweichungen und Neuheiten außerhalb der Trainingsdatenverteilung, die Unterschiede in der Modellkalibrierung ausnutzt (siehe 7.14):** Eine Kalibrierung von ML-Modellen ist typischerweise entscheidend und notwendig für präzise (vertrauenswürdige) Vorhersagen. Wir zeigen zusätzlich auf, dass Variationen und Unterschiede in der Modellkalibrierung zudem auch genutzt werden können, um Anomalien und Neuheiten in den Daten zu erkennen; und somit Ausreißer identifiziert werden können.
- **Entwicklung eines Algorithmus für repräsentatives Sampling in Datenströmen mit variablen Quantilfenstern bei unbekannter (großer) oder unendlicher Datenstromlänge (siehe 7.16):** Repräsentatives Sampling in Datenströmen stellt insbesondere dann eine Herausforderung dar, wenn die endgültige Länge (großer) Datenströme unbekannt oder unendlich ist und Entscheidungen in Echtzeit getroffen werden müssen. In diesem Zusammenhang stellen wir einen Ansatz für Quantilsampling vor, der spezifische mathematische Eigenschaften anstrebt und erfüllt.

# Bibliography

Aalst, Wil MP Van der et al. (2010). "Process mining: a two-step approach to balance between underfitting and overfitting". In: *Software & Systems Modeling* 9.1, pp. 87–111.

Abati, Davide et al. (2019). "Latent space autoregression for novelty detection". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 481–490.

Abdullah, Malak, Alia Madain, and Yaser Jararweh (2022). "ChatGPT: Fundamentals, applications and social impacts". In: *2022 Ninth International Conference on Social Networks Analysis, Management and Security (SNAMS)*. IEEE, pp. 1–8.

Adya, Monica and Fred Collopy (1998). "How effective are neural networks at forecasting and prediction? A review and evaluation". In: *Journal of forecasting* 17.5-6, pp. 481–495.

Aggarwal, Charu C (2006). "On biased reservoir sampling in the presence of stream evolution". In: *Proceedings of the 32nd international conference on Very large data bases*, pp. 607–618.

Aggarwal, Charu C, Alexander Hinneburg, and Daniel A Keim (2001). "On the surprising behavior of distance metrics in high dimensional space". In: *International conference on database theory*. Springer, pp. 420–434.

Agostinelli, Forest et al. (2014). "Learning activation functions to improve deep neural networks". In: *arXiv preprint arXiv:1412.6830*.

agreement, Non disclosure (n.d.). \*by BMW Group. The author, affiliated with BMW Group, and the thesis content are bound by a non-disclosure agreement.

Ahlfors, Lars V (1953). "Complex analysis: an introduction to the theory of analytic functions of one complex variable". In: *New York, London* 177.

Ahmed, Mohiuddin, Raihan Seraj, and Syed Mohammed Shamsul Islam (2020). "The k-means algorithm: A comprehensive survey and performance evaluation". In: *Electronics* 9.8, p. 1295.

Aizenberg, Igor (2011a). *Complex-valued neural networks with multi-valued neurons*. Vol. 353. Springer.

— (2011b). "The Multi-Valued Neuron". In: *Complex-Valued Neural Networks with Multi-Valued Neurons*. Springer, pp. 55–94.

Aizenberg, Igor and Claudio Moraga (2007). "Multilayer feedforward neural network based on multi-valued neurons (MLMVN) and a backpropagation learning algorithm". In: *Soft Computing* 11.2, pp. 169–183.

Aizenberg, Igor, Claudio Moraga, and Dmitriy Paliy (2005). "A feedforward neural network based on multi-valued neurons". In: *Computational intelligence, theory and applications*. Springer, pp. 599–612.

Aizenberg, Igor, Dmitriy Paliy, and Jaakko T Astola (2006). "Multilayer neural network based on multi-valued neurons and the blur identification problem". In: *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. IEEE, pp. 473–480.

Aizenberg, Igor et al. (2016). "Multilayer neural network with multi-valued neurons in time series forecasting of oil production". In: *Neurocomputing* 175, pp. 980–989.

Aizenberg, Naum N and Igor N Aizenberg (1992). "CNN based on multi-valued neuron as a model of associative memory for grey scale images". In: *CNNA'92 Proceedings Second International Workshop on Cellular Neural Networks and Their Applications*. IEEE, pp. 36–41.

Albawi, Saad, Tareq Abed Mohammed, and Saad Al-Zawi (2017). "Understanding of a convolutional neural network". In: *2017 International Conference on Engineering and Technology (ICET)*. Ieee, pp. 1–6.

Almeida, Ana et al. (2023). "Time series big data: a survey on data stream frameworks, analysis and algorithms". In: *Journal of Big Data* 10.1, p. 83.

Almeida, Felipe and Geraldo Xexéo (2019). "Word embeddings: A survey". In: *arXiv preprint arXiv:1901.09069*.

Alyafeai, Zaid, Maged Saeed AlShaibani, and Irfan Ahmad (2020). "A survey on transfer learning in natural language processing". In: *arXiv preprint arXiv:2007.04239*.

Amigó, Enrique et al. (2009). "A comparison of extrinsic clustering evaluation metrics based on formal constraints". In: *Information retrieval* 12.4, pp. 461–486.

Amin, Md Faijul and Kazuyuki Murase (2009). "Single-layered complex-valued neural network for real-valued classification problems". In: *Neurocomputing* 72.4-6, pp. 945–955.

Anderson, Joseph et al. (2014). "The more, the merrier: the blessing of dimensionality for learning large Gaussian mixtures". In: *Conference on Learning Theory*. PMLR, pp. 1135–1164.

Andreu-Perez, Javier et al. (2015). "Big data for health". In: *IEEE journal of biomedical and health informatics* 19.4, pp. 1193–1208.

Andrychowicz, Marcin et al. (2016). "Learning to learn by gradient descent by gradient descent". In: *Advances in neural information processing systems*, pp. 3981–3989.

Andy Coenen, Adam Pearce (2019). "Understanding UMAP". In: *Google PAIR*. URL: <https://pair-code.github.io/understanding-umap/>.

Angelov, Plamen and Eduardo Soares (2020). "Towards explainable deep neural networks (xDNN)". In: *Neural Networks* 130, pp. 185–194.

Arora, Siddhant and Srikanta Bedathur (2020). "On embeddings in relational databases". In: *arXiv preprint arXiv:2005.06437*.

Atrey, Pradeep K et al. (2010). "Multimodal fusion for multimedia analysis: a survey". In: *Multimedia systems* 16.6, pp. 345–379.

Babcock, Brian, Mayur Datar, and Rajeev Motwani (2002). "Sampling from a moving window over streaming data". In: *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 633–634.

Bäck, Thomas and Hans-Paul Schwefel (1993). "An overview of evolutionary algorithms for parameter optimization". In: *Evolutionary computation* 1.1, pp. 1–23.

Bagga, Amit and Breck Baldwin (1998). "Entity-based cross-document coreferencing using the vector space model". In: *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pp. 79–85.

Bakarov, Amir (2018). "A survey of word embeddings evaluation methods". In: *arXiv preprint arXiv:1801.09536*.

Bansal, Rashi, Nishant Gaur, and Shailendra Narayan Singh (2016). "Outlier detection: applications and techniques in data mining". In: *2016 6th International conference-cloud system and big data engineering (Confluence)*. IEEE, pp. 373–377.

Barbedo, Jayme Garcia Arnal (2018). "Impact of dataset size and variety on the effectiveness of deep learning and transfer learning for plant disease classification". In: *Computers and electronics in agriculture* 153, pp. 46–53.

Barlow, Horace B (1989). "Unsupervised learning". In: *Neural computation* 1.3, pp. 295–311.

Bassey, Joshua, Lijun Qian, and Xianfang Li (2021). "A survey of complex-valued neural networks". In: *arXiv preprint arXiv:2101.12249*.

Beaton, Albert E and John W Tukey (1974). "The fitting of power series, meaning polynomials, illustrated on band-spectroscopic data". In: *Technometrics* 16.2, pp. 147–185.

Belhaouari, Samir Brahim et al. (2021). "Unsupervised outlier detection in multidimensional data". In: *Journal of Big Data* 8.1, pp. 1–27.

Bella, Antonio et al. (2010). "Calibration of machine learning models". In: *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*. IGI Global, pp. 128–146.

Bellman, Richard E. (1961). *Adaptive Control Processes: A Guided Tour*. Princeton University Press.

Beltagy, Iz, Matthew E Peters, and Arman Cohan (2020). "Longformer: The long-document transformer". In: *arXiv preprint arXiv:2004.05150*.

Ben-David, Shai and Margareta Ackerman (2008). "Measures of clustering quality: A working set of axioms for clustering". In: *Advances in neural information processing systems* 21.

Ben-Tal, Aharon and Arkadi Nemirovski (1998). "Robust convex optimization". In: *Mathematics of operations research* 23.4, pp. 769–805.

Bendale, Abhijit and Terrance E Boult (2016). "Towards open set deep networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1563–1572.

Bengio, Yoshua, Aaron Courville, and Pascal Vincent (2013). "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8, pp. 1798–1828.

Benvenuto, Nevio and Francesco Piazza (1992). "On the complex backpropagation algorithm". In: *IEEE Transactions on Signal Processing* 40.4, pp. 967–969.

Berkhin, Pavel (2006). "A survey of clustering data mining techniques". In: *Grouping multidimensional data*. Springer, pp. 25–71.

Beyer, Kevin et al. (1999). "When is "nearest neighbor" meaningful?" In: *International conference on database theory*. Springer, pp. 217–235.

Bhanja, Samit and Abhishek Das (2018). "Impact of data normalization on deep neural network for time series forecasting". In: *arXiv preprint arXiv:1812.05519*.

Bhatt, Umang et al. (2020). "Explainable machine learning in deployment". In: *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pp. 648–657.

Biggio, Battista and Fabio Roli (2018). "Wild patterns: Ten years after the rise of adversarial machine learning". In: *Pattern Recognition* 84, pp. 317–331.

Bo, Deyu et al. (2020). "Structural deep clustering network". In: *Proceedings of the web conference 2020*, pp. 1400–1410.

Bonnin, Rodolfo (2017). *Machine Learning for Developers: Uplift your regular applications with the power of statistics, analytics, and machine learning*. Packt Publishing Ltd.

Bordawekar, Rajesh and Oded Shmueli (2017). "Using word embedding to enable semantic queries in relational databases". In: *Proceedings of the 1st Workshop on Data Management for End-to-End Machine Learning*, pp. 1–4.

Bordes, Antoine et al. (2011). "Learning structured embeddings of knowledge bases". In: *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Bordes, Antoine et al. (2013). "Translating embeddings for modeling multi-relational data". In: *Advances in neural information processing systems* 26.

Bottou, Léon (2010). "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of COMPSTAT'2010*. Springer, pp. 177–186.

— (2012). "Stochastic gradient descent tricks". In: *Neural networks: Tricks of the trade*. Springer, pp. 421–436.

Bouchachia, Abdelhamid and Witold Pedrycz (2006). "Data clustering with partial supervision". In: *Data Mining and Knowledge Discovery* 12, pp. 47–78.

Boukerche, Azzedine, Lining Zheng, and Omar Alfandi (2020). "Outlier detection: Methods, models, and classification". In: *ACM Computing Surveys (CSUR)* 53.3, pp. 1–37.

Boyd, Stephen, Stephen P Boyd, and Lieven Vandenberghe (2004). *Convex optimization*. Cambridge university press.

Bramer, Max (2007). *Clustering*. Springer.

Braverman, Vladimir, Rafail Ostrovsky, and Carlo Zaniolo (2009). "Optimal sampling from sliding windows". In: *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 147–156.

Brent, Richard P. (1971). "An algorithm with guaranteed convergence for finding a zero of a function". In: *The Computer Journal* 14.4, pp. 422–425.

Brent, Richard P (2013). *Algorithms for minimization without derivatives*. Courier Corporation.

Bressan, Stephane and Xuesong Lu (2009). "Sampling Data Streams: From Reservoir Sampling to Sliding Window Sampling". In: *The 2009 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*.

Breunig, Markus M et al. (2000). "LOF: identifying density-based local outliers". In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pp. 93–104.

Brodie, Michael L and John Mylopoulos (1986). "Knowledge bases vs databases". In: *On Knowledge Base Management Systems*. Springer, pp. 83–86.

Browne, Michael W (2000). "Cross-validation methods". In: *Journal of mathematical psychology* 44.1, pp. 108–132.

Buchanan, Bruce G (2005). "A (very) brief history of artificial intelligence". In: *Ai Magazine* 26.4, pp. 53–53.

Buragohain, Chiranjeeb and Subhash Suri (Jan. 2009). "Quantiles on Streams". In: DOI: 10.1007/978-0-387-39940-9\_290.

Burkart, Nadia and Marco F Huber (2021). "A survey on the explainability of supervised machine learning". In: *Journal of Artificial Intelligence Research* 70, pp. 245–317.

Bzdok, Danilo, Martin Krzywinski, and Naomi Altman (2018). "Machine learning: supervised methods". In: *Nature methods* 15.1, p. 5.

Cai, Hongyun, Vincent W Zheng, and Kevin Chen-Chuan Chang (2018). "A comprehensive survey of graph embedding: Problems, techniques, and applications". In: *IEEE Transactions on Knowledge and Data Engineering* 30.9, pp. 1616–1637.

Campbell, Murray, A Joseph Hoane Jr, and Feng-hsiung Hsu (2002). "Deep blue". In: *Artificial intelligence* 134.1-2, pp. 57–83.

Campello, Ricardo JGB, Davoud Moulavi, and Jörg Sander (2013). "Density-based clustering based on hierarchical density estimates". In: *Pacific-Asia conference on knowledge discovery and data mining*. Springer, pp. 160–172.

Campello, Ricardo JGB et al. (2015). "Hierarchical density estimates for data clustering, visualization, and outlier detection". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 10.1, pp. 1–51.

Cappelli, Raffaele, Dario Maio, and Davide Maltoni (2000). "Combining fingerprint classifiers". In: *International Workshop on Multiple Classifier Systems*. Springer, pp. 351–361.

Caron, Mathilde et al. (2018). "Deep clustering for unsupervised learning of visual features". In: *Proceedings of the European conference on computer vision (ECCV)*, pp. 132–149.

Caruana, Rich and Alexandru Niculescu-Mizil (2006). "An empirical comparison of supervised learning algorithms". In: *Proceedings of the 23rd international conference on Machine learning*, pp. 161–168.

Cen, Yukuo et al. (2019). "Representation learning for attributed multiplex heterogeneous network". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1358–1368.

Chakraborty, Anirban et al. (2018). "Adversarial attacks and defences: A survey". In: *arXiv preprint arXiv:1810.00069*.

Chandola, Varun, Arindam Banerjee, and Vipin Kumar (2009). "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3, pp. 1–58.

Chandrasekaran, Venkat et al. (2012). "The convex geometry of linear inverse problems". In: *Foundations of Computational mathematics* 12.6, pp. 805–849.

Chao, Min-Te (1982). "A general purpose unequal probability sampling plan". In: *Biometrika* 69.3, pp. 653–656.

Chapelle, Olivier, Bernhard Scholkopf, and Alexander Zien (2009). "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]". In: *IEEE Transactions on Neural Networks* 20.3, pp. 542–542.

Chen, Dongdong, Jiancheng Lv, and Yi Zhang (2017). "Unsupervised multi-manifold clustering by learning deep representation". In: *Workshops at the thirty-first AAAI conference on artificial intelligence*.

Chen, Jinghui et al. (2017). "Outlier detection with autoencoder ensembles". In: *Proceedings of the 2017 SIAM international conference on data mining*. SIAM, pp. 90–98.

Chen, Min, Shiwen Mao, and Yunhao Liu (2014). "Big data: A survey". In: *Mobile networks and applications* 19.2, pp. 171–209.

Chen, Zhao et al. (2018). "Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks". In: *International Conference on Machine Learning*. PMLR, pp. 794–803.

Chen, Zhiwei and Aoqian Zhang (2020). "A survey of approximate quantile computation on large-scale data". In: *IEEE Access* 8, pp. 34585–34597.

Chiang, Leo H, Randy J Pell, and Mary Beth Seasholtz (2003). "Exploring process data with the use of robust outlier detection algorithms". In: *Journal of Process Control* 13.5, pp. 437–449.

Chicco, Davide (2021). "Siamese neural networks: An overview". In: *Artificial Neural Networks*, pp. 73–94.

Choi, Hyunsun, Eric Jang, and Alexander A Alemi (2018). "Waic, but why? generative ensembles for robust anomaly detection". In: *arXiv preprint arXiv:1810.01392*.

Chong, Edwin KP and Stanislaw H Zak (2004). *An introduction to optimization*. John Wiley & Sons.

Choromanska, Anna et al. (2015). "The loss surfaces of multilayer networks". In: *Artificial intelligence and statistics*. PMLR, pp. 192–204.

Choromanski, Krzysztof et al. (2020). "Rethinking attention with performers". In: *arXiv preprint arXiv:2009.14794*.

Chowdhery, Aakanksha et al. (2022). "Palm: Scaling language modeling with pathways". In: *arXiv preprint arXiv:2204.02311*.

Chu, Xiangxiang et al. (2021). "Conditional positional encodings for vision transformers". In: *arXiv preprint arXiv:2102.10882*.

Cios, Krzysztof J et al. (2007). "Unsupervised learning: association rules". In: *Data Mining*. Springer, pp. 289–306.

Clarke, Thomas L (1990). "Generalization of neural networks to the complex plane". In: *1990 IJCNN International Joint Conference on Neural Networks*. IEEE, pp. 435–440.

Coggan, Melanie (2004). "Exploration and exploitation in reinforcement learning". In: *Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University*.

Cordonnier, Jean-Baptiste, Andreas Loukas, and Martin Jaggi (2020). "Multi-head attention: Collaborate instead of concatenate". In: *arXiv preprint arXiv:2006.16362*.

Cormode, Graham et al. (2009). "Forward decay: A practical time decay model for streaming systems". In: *2009 IEEE 25th international conference on data engineering*. IEEE, pp. 138–149.

Cui, Yin et al. (2019). "Class-balanced loss based on effective number of samples". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9268–9277.

DALL-E 3, Microsoft Corporation (2023). *Bing DALL-E 3 Image Creator*. December: 2023. URL: <https://www.bing.com/images/create>.

Dauphin, Yann et al. (2014). "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *arXiv preprint arXiv:1406.2572*.

Din, Salah Ud et al. (2021). "Data stream classification with novel class detection: A review, comparison and challenges". In: *Knowledge and Information Systems* 63, pp. 2231–2276.

Ding, Xuemei et al. (2014). "An experimental evaluation of novelty detection methods". In: *Neurocomputing* 135, pp. 313–327.

Donoho, David L et al. (2000). "High-dimensional data analysis: The curses and blessings of dimensionality". In: *AMS math challenges lecture* 1.2000, p. 32.

Dosovitskiy, Alexey et al. (2020). "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929*.

Draxler, Felix et al. (2018). "Essentially no barriers in neural network energy landscape". In: *International conference on machine learning*. PMLR, pp. 1309–1318.

Du, Mengnan, Ninghao Liu, and Xia Hu (2019). "Techniques for interpretable machine learning". In: *Communications of the ACM* 63.1, pp. 68–77.

Du, Simon et al. (2019). "Gradient descent finds global minima of deep neural networks". In: *International Conference on Machine Learning*. PMLR, pp. 1675–1685.

Dufter, Philipp, Martin Schmitt, and Hinrich Schütze (2022). "Position information in transformers: An overview". In: *Computational Linguistics* 48.3, pp. 733–763.

Duistermaat, JJ et al. (2010). "Taylor expansion in several variables". In: *Distributions: Theory and Applications*, pp. 59–63.

D'Andrea, Alessia, Fernando Ferri, and Patrizia Grifoni (2010). "An overview of methods for virtual social networks analysis". In: *Computational social network analysis*, pp. 3–25.

Eduardo, Simao et al. (2020). "Robust variational autoencoders for outlier detection and repair of mixed-type data". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 4056–4066.

Edwards, Aleksandra et al. (2020). "Go simple and pre-train on domain-specific corpora: On the role of training data for text classification". In: *Proceedings of the 28th international conference on computational linguistics*, pp. 5522–5529.

El Sibai, Rayane et al. (2015). "A performance study of the chain sampling algorithm". In: *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS)*, pp. 487–494. DOI: 10.1109/InteCIS.2015.7397265.

Elhamifar, Ehsan and René Vidal (2011). "Sparse manifold clustering and embedding". In: *Advances in neural information processing systems* 24.

Eskin, Eleazar et al. (2002). "A geometric framework for unsupervised anomaly detection". In: *Applications of data mining in computer security*. Springer, pp. 77–101.

Estivill-Castro, Vladimir (2002). "Why so many clustering algorithms: a position paper". In: *ACM SIGKDD explorations newsletter* 4.1, pp. 65–75.

Everett, B (2013). *An introduction to latent variable models*. Springer Science & Business Media.

Fan, Angela, Mike Lewis, and Yann Dauphin (2018). "Hierarchical neural story generation". In: *arXiv preprint arXiv:1805.04833*.

Fang, Ziquan et al. (2022). "Spatio-temporal trajectory similarity learning in road networks". In: *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 347–356.

Fei, Geli and Bing Liu (2016). "Breaking the closed world assumption in text classification". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 506–514.

Feinberg, Eugene A and Adam Shwartz (2012). *Handbook of Markov decision processes: methods and applications*. Vol. 40. Springer Science & Business Media.

Feldman, Ronen, James Sanger, et al. (2007). *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge university press.

Finch, Tony (2009). "Incremental calculation of weighted mean and variance". In: URL: <https://fanf2.user.srccf.net/hermes/doc/antiforgery/stats.pdf>.

Finn, Chelsea et al. (2016). "Generalizing skills with semi-supervised reinforcement learning". In: *arXiv preprint arXiv:1612.00429*.

Foorthuis, Ralph (2018). "A typology of data anomalies". In: *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*. Springer, pp. 26–38.

Foret, Pierre et al. (2020). "Sharpness-aware minimization for efficiently improving generalization". In: *arXiv preprint arXiv:2010.01412*.

Fowlkes, Edward B and Colin L Mallows (1983). "A method for comparing two hierarchical clusterings". In: *Journal of the American statistical association* 78.383, pp. 553–569.

Frey, Brendan J and Delbert Dueck (2007). "Clustering by passing messages between data points". In: *science* 315.5814, pp. 972–976.

Fu, Xinyu et al. (2020). "Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding". In: *Proceedings of The Web Conference 2020*, pp. 2331–2341.

Fu, Yao et al. (2022). "Complexity-based prompting for multi-step reasoning". In: *arXiv preprint arXiv:2210.00720*.

Gaber, Mohamed Medhat, Arkady Zaslavsky, and Shonali Krishnaswamy (2007). "A survey of classification methods in data streams". In: *Data Streams: models and algorithms*, pp. 39–59.

García, Salvador, Julián Luengo, and Francisco Herrera (2015). *Data preprocessing in data mining*. Vol. 72. Springer.

García, Salvador et al. (2016). "Big data preprocessing: methods and prospects". In: *Big Data Analytics* 1.1, pp. 1–22.

Gawlikowski, Jakob et al. (2021). "A survey of uncertainty in deep neural networks". In: *arXiv preprint arXiv:2107.03342*.

Ge, Weifeng (2018). "Deep metric learning with hierarchical triplet loss". In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 269–285.

Ge, ZongYuan et al. (2017). "Generative openmax for multi-class open set classification". In: *arXiv preprint arXiv:1707.07418*.

Gelman, Andrew et al. (1995). *Bayesian data analysis*. Chapman and Hall/CRC.

Ghahramani, Zoubin (2003). "Unsupervised learning". In: *Summer School on Machine Learning*. Springer, pp. 72–112.

Ghojogh, Benyamin and Ali Ghodsi (2020). "Attention mechanism, transformers, BERT, and GPT: tutorial and survey". In.

Golan, Izhak and Ran El-Yaniv (2018). "Deep anomaly detection using geometric transformations". In: *Advances in neural information processing systems 31*.

Goldberg, Yoav and Omer Levy (2014). "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method". In: *arXiv preprint arXiv:1402.3722*.

Gontijo-Lopes, Raphael et al. (2020). "Affinity and diversity: Quantifying mechanisms of data augmentation". In: *arXiv preprint arXiv:2002.08973*.

Goodfellow, Ian et al. (2013). "Maxout networks". In: *International conference on machine learning*. PMLR, pp. 1319–1327.

Goodfellow, Ian et al. (2016). *Deep learning*. Vol. 1. MIT press Cambridge.

Gorban, Alexander N and Ivan Yu Tyukin (2018). "Blessing of dimensionality: mathematical foundations of the statistical physics of data". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 376.2118, p. 20170237.

Goyal, Palash and Emilio Ferrara (2018). "Graph embedding techniques, applications, and performance: A survey". In: *Knowledge-Based Systems* 151, pp. 78–94.

Groenendijk, Rick et al. (2021). "Multi-loss weighting with coefficient of variations". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1469–1478.

Gruhl, Christian, Bernhard Sick, and Sven Tomforde (2021). "Novelty detection in continuously changing environments". In: *Future Generation Computer Systems* 114, pp. 138–154.

Gu, Albert and Tri Dao (2023). "Mamba: Linear-time sequence modeling with selective state spaces". In: *arXiv preprint arXiv:2312.00752*.

Gu, Albert, Karan Goel, and Christopher Ré (2021). "Efficiently modeling long sequences with structured state spaces". In: *arXiv preprint arXiv:2111.00396*.

Guimerà Cuevas, Felip and Thomy Phan (2021). *Complex Value-based Deep Learning and Applications to Reinforcement Learning*. Projektarbeits-Abschlussvortrag. Supervisors: Thomy Phan, Thomas Gabor. URL: <https://www.mobile.ifi.lmu.de/lehre/oberseminar/?date=2021-04-08>.

Guimerà Cuevas, Felip, Thomy Phan, and Helmut Schmid (2023). "Adaptive Bi-nonlinear Neural Networks Based on Complex Numbers with Weights Constrained Along the Unit Circle". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, pp. 355–366.

Guimerà Cuevas, Felip and Helmut Schmid (n.d.). " $\alpha$ Max-B-CUBED: A Supervised Metric for Addressing Completeness and Uncertainty in Cluster Evaluation". In: () .

— (2024a). "Calibration Misalignment as a Post-hoc Approach for Out-of-Distribution Detection in Deep Neural Networks". In: *Proceedings of the International Conference on Pattern Recognition and Artificial Intelligence (ICPR-AI)*. URL: <https://brain.korea.ac.kr/icprai2024/acceptedpapers.php>.

— (2024b). "Robust Non-linear Normalization of Heterogeneous Feature Distributions with Adaptive Tanh-Estimators". In: *The 27th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Accepted and scheduled for publication after the conference proceedings (<https://aistats.org>).

Gunning, David et al. (2019). "XAI—Explainable artificial intelligence". In: *Science robotics* 4.37, eaay7120.

Guo, Chuan et al. (2017). "On calibration of modern neural networks". In: *International conference on machine learning*. PMLR, pp. 1321–1330.

Gurney, Kevin (2018). *An introduction to neural networks*. CRC press.

Haas, Peter J (2016). "Data-stream sampling: Basic techniques and results". In: *Data Stream Management: Processing High-Speed Data Streams*, pp. 13–44.

Hadi, Muhammad Usman et al. (2023). "A survey on large language models: Applications, challenges, limitations, and practical usage". In: *TechRxiv*.

Hammersley, John (2013). *Monte carlo methods*. Springer Science & Business Media.

Hampel, Frank R (1973). "Robust estimation: A condensed partial survey". In: *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete* 27.2, pp. 87–104.

— (1974). "The influence curve and its role in robust estimation". In: *Journal of the american statistical association* 69.346, pp. 383–393.

Hampel, Frank R et al. (2011). *Robust statistics: the approach based on influence functions*. Vol. 196. John Wiley & Sons.

Han, Songqiao et al. (2022). "Adbench: Anomaly detection benchmark". In: *Advances in Neural Information Processing Systems* 35, pp. 32142–32159.

Hand, David J (2007). "Principles of data mining". In: *Drug safety* 30.7, pp. 621–622.

Hasani, Ramin et al. (2021). "Liquid time-constant networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 9, pp. 7657–7666.

Haugeland, John (1997). *Mind design II: philosophy, psychology, artificial intelligence*. MIT press.

Hawkins, Douglas M (2004). "The problem of overfitting". In: *Journal of chemical information and computer sciences* 44.1, pp. 1–12.

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Head, Cari Beth et al. (2023). "Large language model applications for evaluation: Opportunities and ethical implications". In: *New Directions for Evaluation* 2023.178–179, pp. 33–46.

Healy, Kieran (2018). *Data visualization: a practical introduction*. Princeton University Press.

Hessel, Jack et al. (2021). "Clipscore: A reference-free evaluation metric for image captioning". In: *arXiv preprint arXiv:2104.08718*.

Heusden, Ruben van, Jaap Kamps, and Maarten Marx (2022). "BCubed Revisited: Elements Like Me". In: *Proceedings of the 2022 ACM SIGIR International Conference on Theory of Information Retrieval*, pp. 127–132.

Hinkley, David V (1973). *Robust Estimates of Location: Survey and Advances*.

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean (2015). "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531*.

Hinton, Geoffrey E and Ruslan R Salakhutdinov (2006). "Reducing the dimensionality of data with neural networks". In: *science* 313.5786, pp. 504–507.

Hirose, Akira (1992). "Dynamics of fully complex-valued neural networks". In: *Electronics letters* 28.16, pp. 1492–1494.

— (2003). *Complex-valued neural networks: theories and applications*. Vol. 5. World Scientific.

Hirose, Akira (2012). *Complex-valued neural networks*. Vol. 400. Springer Science & Business Media.

Hjaltason, Gisli R. and Hanan Samet (2003). "Properties of embedding methods for similarity searching in metric spaces". In: *IEEE Transactions on Pattern Analysis and machine intelligence* 25.5, pp. 530–549.

Holtzman, Ari et al. (2019). "The curious case of neural text degeneration". In: *arXiv preprint arXiv:1904.09751*.

Hong, Huiting et al. (2020). "An attention-based graph neural network for heterogeneous structural learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 04, pp. 4132–4139.

Hotho, Andreas, Andreas Nürnberger, and Gerhard Paß (2005). "A brief survey of text mining." In: *Ldv Forum*. Vol. 20. 1. Citeseer, pp. 19–62.

Hsu, Yen-Chang et al. (2020). "Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10951–10960.

Hu, Renjun et al. (2016). "An embedding approach to anomaly detection". In: *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, pp. 385–396.

Hu, Ziniu et al. (2020). "Heterogeneous graph transformer". In: *Proceedings of the web conference 2020*, pp. 2704–2710.

Huang, Peihao et al. (2014). "Deep embedding network for clustering". In: *2014 22nd International conference on pattern recognition*. IEEE, pp. 1532–1537.

Huang, Xiaowei et al. (2017). "Safety verification of deep neural networks". In: *International conference on computer aided verification*. Springer, pp. 3–29.

Hunt, Kenneth J et al. (1992). "Neural networks for control systems—a survey". In: *Automatica* 28.6, pp. 1083–1112.

Indyk, Piotr and Rajeev Motwani (1998). "Approximate nearest neighbors: towards removing the curse of dimensionality". In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613.

Ioffe, Sergey and Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR, pp. 448–456.

Izmailov, Pavel et al. (2018). "Averaging weights leads to wider optima and better generalization". In: *arXiv preprint arXiv:1803.05407*.

Jabbar, H and Rafiqul Zaman Khan (2015). "Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study)". In: *Computer Science, Communication and Instrumentation Devices*, pp. 163–172.

Jain, Anil, Karthik Nandakumar, and Arun Ross (2005). "Score normalization in multimodal biometric systems". In: *Pattern recognition* 38.12, pp. 2270–2285.

Jaiswal, Ashish et al. (2021). "A survey on contrastive self-supervised learning". In: *Technologies* 9.1, p. 2.

Jang, Eric, Shixiang Gu, and Ben Poole (2016). "Categorical reparameterization with gumbel-softmax". In: *arXiv preprint arXiv:1611.01144*.

Jayalakshmi, T and A Santhakumaran (2011). "Statistical normalization and back propagation for classification". In: *International Journal of Computer Theory and Engineering* 3.1, pp. 1793–8201.

Jayaram, Rajesh, David P Woodruff, and Samson Zhou (2022). "Truly perfect samplers for data streams and sliding windows". In: *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pp. 29–40.

Kaddour, Jean et al. (2023). "Challenges and applications of large language models". In: *arXiv preprint arXiv:2307.10169*.

Kaelbling, Leslie Pack, Michael L Littman, and Andrew W Moore (1996). "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4, pp. 237–285.

Kainen, Paul C (1997). "Utilizing geometric anomalies of high dimension: When complexity makes computation easier". In: *Computer Intensive Methods in Control and Signal Processing*. Springer, pp. 283–294.

Kalyan, Katikapalli Subramanyam, Ajit Rajasekharan, and Sivanesan Sangeetha (2021). "Ammus: A survey of transformer-based pretrained models in natural language processing". In: *arXiv preprint arXiv:2108.05542*.

Kapil, Gayatri, Alka Agrawal, and RA Khan (2016). "A study of big data characteristics". In: *2016 International Conference on Communication and Electronics Systems (ICCES)*. IEEE, pp. 1–4.

Ke, Guolin, Di He, and Tie-Yan Liu (2020). "Rethinking positional encoding in language pre-training". In: *arXiv preprint arXiv:2006.15595*.

Ke, Zhanghan et al. (2019). "Dual student: Breaking the limits of the teacher in semi-supervised learning". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6728–6736.

Kendall, Alex, Yarin Gal, and Roberto Cipolla (2018). "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7482–7491.

Khan, Arif Ali et al. (2021a). "Ethics of AI: A Systematic Literature Review of Principles and Challenges". In: *arXiv preprint arXiv:2109.07906*.

Khan, Dost Muhammad et al. (2021b). "A New Efficient Redescending M-Estimator for Robust Fitting of Linear Regression Models in the Presence of Outliers". In: *Mathematical Problems in Engineering* 2021.

Khan, Salman et al. (2021c). "Transformers in vision: A survey". In: *arXiv preprint arXiv:2101.01169*.

Khanum, Memoona et al. (2015). "A survey on unsupervised machine learning algorithms for automation, classification and maintenance". In: *International Journal of Computer Applications* 119.13.

King, Gary and Langche Zeng (2001). "Logistic regression in rare events data". In: *Political analysis* 9.2, pp. 137–163.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Kingma, Durk P, Tim Salimans, and Max Welling (2015). "Variational dropout and the local reparameterization trick". In: *Advances in neural information processing systems* 28.

Kitaev, Nikita, Łukasz Kaiser, and Anselm Levskaya (2020). "Reformer: The efficient transformer". In: *arXiv preprint arXiv:2001.04451*.

Klambauer, Günter et al. (2017). "Self-normalizing neural networks". In: *Advances in neural information processing systems* 30.

Kleinberg, Jon (2002). "An impossibility theorem for clustering". In: *Advances in neural information processing systems* 15.

Koci, Tom and Ondřej Bojar (2017). "An exploration of word embedding initialization in deep-learning tasks". In: *arXiv preprint arXiv:1711.09160*.

Koehrsen, Will (2018). "Overfitting vs. underfitting: A complete example". In: *Towards Data Science*.

Kolajo, Taiwo, Olawande Daramola, and Ayodele Adebiyi (2019). "Big data stream analysis: a systematic literature review". In: *Journal of Big Data* 6.1, p. 47.

Komić, Jasmin (2011). "Harmonic Mean". In: *International Encyclopedia of Statistical Science*. Ed. by Miodrag Lovric. Berlin, Heidelberg: Springer Berlin Heidelberg,

pp. 622–624. ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2\_645.  
URL: [https://doi.org/10.1007/978-3-642-04898-2\\_645](https://doi.org/10.1007/978-3-642-04898-2_645).

Köppen, Mario (2000). "The curse of dimensionality". In: *5th Online World Conference on Soft Computing in Industrial Applications (WSC5)*. Vol. 1, pp. 4–8.

Kosub, Sven (2019). "A note on the triangle inequality for the Jaccard distance". In: *Pattern Recognition Letters* 120, pp. 36–38.

Krause, Jonathan et al. (2017). "A Hierarchical Approach for Generating Descriptive Image Paragraphs". In: *Computer Vision and Pattern Recognition (CVPR)*.

Kriegel, Hans-Peter et al. (2011). "Interpreting and unifying outlier scores". In: *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM, pp. 13–24.

Krizhevsky, Alex, Geoffrey Hinton, et al. (2009). "Learning multiple layers of features from tiny images". In.

Kuhn, Max, Kjell Johnson, et al. (2013). *Applied predictive modeling*. Vol. 26. Springer.

Kull, Meelis, Telmo Silva Filho, and Peter Flach (2017). "Beta calibration: a well-founded and easily implemented improvement on logistic calibration for binary classifiers". In: *Artificial Intelligence and Statistics*. PMLR, pp. 623–631.

Kull, Meelis et al. (2019). "Beyond temperature scaling: Obtaining well-calibrated multi-class probabilities with dirichlet calibration". In: *Advances in neural information processing systems* 32.

Kuroe, Yasuaki, Mitsuo Yoshida, and Takehiro Mori (2003). "On activation functions for complex-valued neural networks—existence of energy functions—". In: *Artificial Neural Networks and Neural Information Processing—ICANN/ICONIP 2003*. Springer, pp. 985–992.

Laine, Samuli and Timo Aila (2016). "Temporal ensembling for semi-supervised learning". In: *arXiv preprint arXiv:1610.02242*.

Latha, L and S Thangasamy (2011). "Efficient approach to normalization of multi-modal biometric scores". In: *International Journal of Computer Applications* 32.10, pp. 57–64.

Learning, Semi-Supervised (2006). "Semi-Supervised Learning". In: *CSZ2006.html*.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *nature* 521.7553, pp. 436–444.

LeCun, Yann et al. (1989). "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4, pp. 541–551.

LeCun, Yann A et al. (2012). "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, pp. 9–48.

Lee, John A and Michel Verleysen (2007). *Nonlinear dimensionality reduction*. Springer Science & Business Media.

Leofante, Francesco et al. (2018). "Automated verification of neural networks: Advances, challenges and perspectives". In: *arXiv preprint arXiv:1805.09938*.

Leonowicz, Zbigniew, Juha Karvanen, and Sergei L Shishkin (2005). "Trimmed estimators for robust averaging of event-related potentials". In: *Journal of neuroscience methods* 142.1, pp. 17–26.

Lester, Brian, Rami Al-Rfou, and Noah Constant (2021). "The power of scale for parameter-efficient prompt tuning". In: *arXiv preprint arXiv:2104.08691*.

Leung, Henry and Simon Haykin (1991). "The complex backpropagation algorithm". In: *IEEE Transactions on signal processing* 39.9, pp. 2101–2104.

Leusin, Matheus E., Bjoern Jindra, and Daniel S. Hain (2021). "An evolutionary view on the emergence of Artificial Intelligence". In: *arXiv preprint arXiv:2102.00233*.

Li, Boyi et al. (2021a). "On feature normalization and data augmentation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12383–12392.

Li, Chaoqun and Hongwei Li (2010). "A Survey of Distance Metrics for Nominal Attributes." In: *J. Softw.* 5.11, pp. 1262–1269.

Li, Fan, Nick Ruijs, and Yuan Lu (2023). "Ethics AI: A Systematic Review on Ethical Concerns and Related Strategies for Designing with AI in Healthcare". In: *AI* 4.1, pp. 28–53. DOI: 10.3390/ai4010003.

Li, Hao et al. (2018a). "Visualizing the loss landscape of neural nets". In: *Advances in neural information processing systems* 31.

Li, Minghao et al. (2021b). "Trocr: Transformer-based optical character recognition with pre-trained models". In: *arXiv preprint arXiv:2109.10282*.

Li, Xiang Lisa and Percy Liang (2021). "Prefix-tuning: Optimizing continuous prompts for generation". In: *arXiv preprint arXiv:2101.00190*.

Li, Xiucheng et al. (2018b). "Deep representation learning for trajectory similarity computation". In: *2018 IEEE 34th international conference on data engineering (ICDE)*. IEEE, pp. 617–628.

Li, Yujia et al. (2015). "Gated graph sequence neural networks". In: *arXiv preprint arXiv:1511.05493*.

Li, Yuxi (2017). "Deep reinforcement learning: An overview". In: *arXiv preprint arXiv:1701.07274*.

Li, Zengyi et al. (2022). "Neural manifold clustering and embedding". In: *arXiv preprint arXiv:2201.10000*.

Liang, Chunquan, Mei Li, and Bin Liu (2019). "Online computing quantile summaries over uncertain data streams". In: *IEEE Access* 7, pp. 10916–10926.

Liang, Shiyu, Yixuan Li, and Rayadurgam Srikant (2017). "Enhancing the reliability of out-of-distribution image detection in neural networks". In: *arXiv preprint arXiv:1706.02690*.

Lillicrap, Timothy P and Adam Santoro (2019). "Backpropagation through time and the brain". In: *Current opinion in neurobiology* 55, pp. 82–89.

Lillicrap, Timothy P et al. (2020). "Backpropagation and the brain". In: *Nature Reviews Neuroscience* 21.6, pp. 335–346.

Lin, Tianyang et al. (2022). "A survey of transformers". In: *AI Open*.

Lin, Tsung-Yi et al. (2014). "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer, pp. 740–755.

Lin, Tsung-Yi et al. (2017). "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988.

Lin, X. et al. (2004). "Continuously maintaining quantile summaries of the most recent N elements over a data stream". In: *Proceedings. 20th International Conference on Data Engineering*, pp. 362–373. DOI: 10.1109/ICDE.2004.1320011.

Little, Gordon R, Steven C Gustafson, and Robert A Senn (1990). "Generalization of the backpropagation neural network learning algorithm to permit complex weights". In: *Applied Optics* 29.11, pp. 1591–1592.

Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou (2008). "Isolation forest". In: *2008 eighth ieee international conference on data mining*. IEEE, pp. 413–422.

Liu, Weibo et al. (2017). "A survey of deep neural network architectures and their applications". In: *Neurocomputing* 234, pp. 11–26.

Liu, Wen et al. (2018). "Future frame prediction for anomaly detection—a new baseline". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6536–6545.

Liu, Xiao et al. (2023). "GPT understands, too". In: *AI Open*.

Liu, Ze et al. (2021). "Swin transformer: Hierarchical vision transformer using shifted windows". In: *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 10012–10022.

Liu, Ze et al. (2022). "Swin transformer v2: Scaling up capacity and resolution". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12009–12019.

Loshchilov, Ilya and Frank Hutter (2017). "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101*.

Lu, Yao et al. (2021). "Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity". In: *arXiv preprint arXiv:2104.08786*.

Lukasik, Michal et al. (2020). "Does label smoothing mitigate label noise?" In: *International Conference on Machine Learning*. PMLR, pp. 6448–6458.

Lundberg, Scott M and Su-In Lee (2017). "A unified approach to interpreting model predictions". In: *Advances in neural information processing systems* 30.

Lupea, Valentin Mircea (2012). "Multi-Valued Neuron with a periodic activation function—New learning strategy". In: *2012 IEEE 8th International Conference on Intelligent Computer Communication and Processing*. IEEE, pp. 79–82.

Lheureux, Alexandra et al. (2017). "Machine learning with big data: Challenges and approaches". In: *Ieee Access* 5, pp. 7776–7797.

Ma, Xutai et al. (2019). "Monotonic multihead attention". In: *arXiv preprint arXiv:1909.12406*.

Madhulatha, T Soni (2012). "An overview on clustering methods". In: *arXiv preprint arXiv:1205.1117*.

Magdy, Nehal et al. (2015). "Review on trajectory similarity measures". In: *2015 IEEE seventh international conference on Intelligent Computing and Information Systems (ICICIS)*. IEEE, pp. 613–619.

Markou, Markos and Sameer Singh (2003). "Novelty detection: a review—part 1: statistical approaches". In: *Signal processing* 83.12, pp. 2481–2497.

Marx, Vivien (2013). "The big challenges of big data". In: *Nature* 498.7453, pp. 255–260.

Mayer, Ruben and Hans-Arno Jacobsen (2020). "Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools". In: *ACM Computing Surveys (CSUR)* 53.1, pp. 1–37.

McAfee, Andrew et al. (2012). "Big data: the management revolution". In: *Harvard business review* 90.10, pp. 60–68.

McCarthy, John (1998). "What is artificial intelligence?" In.

McConville, Ryan et al. (2021). "N2d:(not too) deep clustering via clustering the local manifold of an autoencoded embedding". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, pp. 5145–5152.

McInnes, Leland, John Healy, and Steve Astels (2017a). "hdbscan: Hierarchical density based clustering." In: *J. Open Source Softw.* 2.11, p. 205.

— (2017b). "hdbscan: Hierarchical density based clustering". In: *The Journal of Open Source Software* 2.11. DOI: 10.21105/joss.00205. URL: <https://doi.org/10.21105/joss.00205>.

McInnes, Leland, John Healy, and James Melville (2018). "Umap: Uniform manifold approximation and projection for dimension reduction". In: *arXiv preprint arXiv:1802.03426*.

McKay, Michael D, Richard J Beckman, and William J Conover (2000). "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code". In: *Technometrics* 42.1, pp. 55–61.

Miller, James N (1993). "Tutorial review—Outliers in experimental data and their treatment". In: *Analyst* 118.5, pp. 455–461.

Minsker, Stanislav (2015). "Geometric median and robust estimation in Banach spaces". In: *Bernoulli* 21.4, pp. 2308–2335.

Minsky, Marvin (1961). "Steps toward artificial intelligence". In: *Proceedings of the IRE* 49.1, pp. 8–30.

Mises, RV and Hilda Pollaczek-Geiringer (1929). "Praktische Verfahren der Gleichungsauflösung." In: *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik* 9.1, pp. 58–77.

Miyato, Takeru et al. (2018). "Spectral normalization for generative adversarial networks". In: *arXiv preprint arXiv:1802.05957*.

Molnar, Christoph (2020). *Interpretable machine learning*. Lulu. com.

Moreno, Jose G and Gaël Dias (2015). "Adapted b-cubed metrics to unbalanced datasets". In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 911–914.

Morningstar, Warren et al. (2021). "Density of states estimation for out of distribution detection". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 3232–3240.

Mu, Norman and Justin Gilmer (2019). "Mnist-c: A robustness benchmark for computer vision". In: *arXiv preprint arXiv:1906.02337*.

Muandet, Krikamol et al. (2016). "Kernel mean embedding of distributions: A review and beyond". In: *arXiv preprint arXiv:1605.09522*.

Muandet, Krikamol et al. (2017). "Kernel mean embedding of distributions: A review and beyond". In: *Foundations and Trends® in Machine Learning* 10.1-2, pp. 1–141.

Muhammad, Iqbal and Zhu Yan (2015). "SUPERVISED MACHINE LEARNING APPROACHES: A SURVEY." In: *ICTACT Journal on Soft Computing* 5.3.

Müller, Rafael, Simon Kornblith, and Geoffrey Hinton (2019). "When does label smoothing help?" In: *arXiv preprint arXiv:1906.02629*.

Murtagh, Fionn and Pedro Contreras (2012). "Algorithms for hierarchical clustering: an overview". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2.1, pp. 86–97.

Nalisnick, Eric et al. (2018). "Do deep generative models know what they don't know?" In: *arXiv preprint arXiv:1810.09136*.

Nandakumar, Karthik et al. (2007). "Likelihood ratio-based biometric score fusion". In: *IEEE transactions on pattern analysis and machine intelligence* 30.2, pp. 342–347.

Neishi, Masato et al. (2017). "A bag of useful tricks for practical neural machine translation: Embedding layer initialization and large batch size". In: *Proceedings of the 4th Workshop on Asian Translation (WAT2017)*, pp. 99–109.

Newman, Mark EJ, Duncan J Watts, and Steven H Strogatz (2002). "Random graph models of social networks". In: *Proceedings of the national academy of sciences* 99.suppl 1, pp. 2566–2572.

Nickel, Maximilian, Lorenzo Rosasco, and Tomaso Poggio (2016). "Holographic embeddings of knowledge graphs". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1.

Nickel, Maximilian et al. (2015). "A review of relational machine learning for knowledge graphs". In: *Proceedings of the IEEE* 104.1, pp. 11–33.

Nilsson, Nils J (2014). *Principles of artificial intelligence*. Morgan Kaufmann.

Nitta, Tohru (2004). "Orthogonality of decision boundaries in complex-valued neural networks". In: *Neural computation* 16.1, pp. 73–97.

Niu, Zhaoyang, Guoqiang Zhong, and Hui Yu (2021). "A review on the attention mechanism of deep learning". In: *Neurocomputing* 452, pp. 48–62.

Oord, Aaron van den, Yazhe Li, and Oriol Vinyals (2018). "Representation learning with contrastive predictive coding". In: *arXiv preprint arXiv:1807.03748*.

OpenAI (2023). *GPT-4 Technical Report*. arXiv: 2303.08774 [cs.CL].

Ouali, Yassine, Céline Hudelot, and Myriam Tami (2020). “An overview of deep semi-supervised learning”. In: *arXiv preprint arXiv:2006.05278*.

Paleyes, Andrei, Raoul-Gabriel Urma, and Neil D Lawrence (2022). “Challenges in deploying machine learning: a survey of case studies”. In: *ACM Computing Surveys* 55.6, pp. 1–29.

Panaretos, Victor M and Yoav Zemel (2019). “Statistical aspects of Wasserstein distances”. In: *Annual review of statistics and its application* 6, pp. 405–431.

Pang, Guansong et al. (2021). “Deep learning for anomaly detection: A review”. In: *ACM computing surveys (CSUR)* 54.2, pp. 1–38.

Papadimitriou, Spiros et al. (2003). “Loci: Fast outlier detection using the local correlation integral”. In: *Proceedings 19th international conference on data engineering (Cat. No. 03CH37405)*. IEEE, pp. 315–326.

Parmar, Jitendra et al. (2023). “Open-world machine learning: applications, challenges, and opportunities”. In: *ACM Computing Surveys* 55.10, pp. 1–37.

Parsons, Van L (2014). “Stratified sampling”. In: *Wiley StatsRef: Statistics Reference Online*, pp. 1–11.

Paszke, Adam et al. (2017). “Automatic differentiation in pytorch”. In.

Penedo, Guilherme et al. (2023). “The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only”. In: *arXiv preprint arXiv:2306.01116*.

Pennington, Jeffrey, Richard Socher, and Christopher D Manning (2014). “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Perera, Pramuditha and Vishal M Patel (2019). “Deep transfer learning for multiple class novelty detection”. In: *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, pp. 11544–11552.

Perez, Luis and Jason Wang (2017). “The effectiveness of data augmentation in image classification using deep learning”. In: *arXiv preprint arXiv:1712.04621*.

Pfahringer, Bernhard (2017). “Disjunctive Normal Form”. In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer.

Philipp, George, Dawn Song, and Jaime G Carbonell (2017). “The exploding gradient problem demystified-definition, prevalence, impact, origin, tradeoffs, and solutions”. In: *arXiv preprint arXiv:1712.05577*.

Phillips, Jeff M and Suresh Venkatasubramanian (2011). “A gentle introduction to the kernel distance”. In: *arXiv preprint arXiv:1103.1625*.

Pimentel, Marco AF et al. (2014). “A review of novelty detection”. In: *Signal processing* 99, pp. 215–249.

Plastria, Frank (2011). “The Weiszfeld algorithm: proof, amendments, and extensions”. In: *Foundations of location analysis*. Springer, pp. 357–389.

Platt, John et al. (1999). “Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods”. In: *Advances in large margin classifiers* 10.3, pp. 61–74.

Press, William H et al. (2007). *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

Prince, S. (June 2022). *Out-of-distribution detection I: anomaly detection*. Borealis AI. URL: <https://www.borealisai.com/research-blogs/out-distribution-detection-i-anomaly-detection/>.

Pukelsheim, Friedrich (1994). “The three sigma rule”. In: *The American Statistician* 48.2, pp. 88–91.

Qian, Ning (1999). "On the momentum term in gradient descent learning algorithms". In: *Neural networks* 12.1, pp. 145–151.

Qian, Sheng et al. (2018). "Adaptive activation functions in convolutional neural networks". In: *Neurocomputing* 272, pp. 204–212.

Radford, Alec et al. (2019). "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8, p. 9.

Radford, Alec et al. (2021). "Learning transferable visual models from natural language supervision". In: *International conference on machine learning*. PMLR, pp. 8748–8763.

Rai, Pradeep and Shubha Singh (2010). "A survey of clustering techniques". In: *International Journal of Computer Applications* 7.12, pp. 1–5.

Ramdas, Aaditya, Nicolás García Trillo, and Marco Cuturi (2017). "On wasserstein two-sample testing and related families of nonparametric tests". In: *Entropy* 19.2, p. 47.

Rand, William M (1971). "Objective criteria for the evaluation of clustering methods". In: *Journal of the American Statistical association* 66.336, pp. 846–850.

Rao, Singiresu S (2019). *Engineering optimization: theory and practice*. John Wiley & Sons.

Rebrickable® (2022). *Rebrickable® Database*. URL: <https://rebrickable.com/downloads/> (visited on 09/21/2022).

Ren, Jie et al. (2019). "Likelihood ratios for out-of-distribution detection". In: *Advances in neural information processing systems* 32.

Reuther, Albert et al. (2020). "Survey of machine learning accelerators". In: *2020 IEEE high performance extreme computing conference (HPEC)*. IEEE, pp. 1–12.

Reynolds, Douglas A et al. (2009). "Gaussian mixture models". In: *Encyclopedia of biometrics* 741.659–663.

Ribaric, Slobodan and Ivan Fratric (2005). "A matching-score normalization technique for multimodal biometric systems". In: *Proceedings of Third COST 275 Workshop-Biometrics on the Internet*. University of Hertfordshire, pp. 55–58.

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2016). "" Why should i trust you?" Explaining the predictions of any classifier". In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144.

Riedmiller, Martin and Heinrich Braun (1993). "A direct adaptive method for faster backpropagation learning: The RPROP algorithm". In: *IEEE international conference on neural networks*. IEEE, pp. 586–591.

Rillig, Matthias C et al. (2023). "Risks and benefits of large language models for the environment". In: *Environmental Science & Technology* 57.9, pp. 3464–3466.

Roberts, Arthur Wayne (1993). "Convex functions". In: *Handbook of Convex Geometry*. Elsevier, pp. 1081–1104.

Rodriguez, Alex and Alessandro Laio (2014). "Clustering by fast search and find of density peaks". In: *science* 344.6191, pp. 1492–1496.

Rokach, Lior and Oded Maimon (2005). "Clustering methods". In: *Data mining and knowledge discovery handbook*. Springer, pp. 321–352.

Rosales-Méndez, Henry and Yunior Ramírez-Cruz (2013). "Cice-bcubed: A new evaluation measure for overlapping clustering algorithms". In: *Iberoamerican Congress on Pattern Recognition*. Springer, pp. 157–164.

Rosenblatt, Frank (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory.

Rousseeuw, Peter J and Mia Hubert (2011). "Robust statistics for outlier detection". In: *Wiley interdisciplinary reviews: Data mining and knowledge discovery* 1.1, pp. 73–79.

Ruder, Sebastian (2016). "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747*.

Ruff, Lukas et al. (2018). "Deep One-Class Classification". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 4393–4402. URL: <https://proceedings.mlr.press/v80/ruff18a.html>.

Rumelhart, David E et al. (1995). "Backpropagation: The basic theory". In: *Backpropagation: Theory, architectures and applications*, pp. 1–34.

Rüschendorf, Ludger (1985). "The Wasserstein distance and approximation theorems". In: *Probability Theory and Related Fields* 70.1, pp. 117–129.

Ruspini, Enrique H (1969). "A new approach to clustering". In: *Information and control* 15.1, pp. 22–32.

Russell, Stuart and Peter Norvig (2002). "Artificial intelligence: a modern approach". In.

Saerens, Marco, Patrice Latinne, and Christine Decaestecker (2002). "Adjusting the outputs of a classifier to new a priori probabilities: a simple procedure". In: *Neural computation* 14.1, pp. 21–41.

Saisubramanian, Sandhya, Sainyam Galhotra, and Shlomo Zilberstein (2020). "Balancing the tradeoff between clustering value and interpretability". In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pp. 351–357.

Sajjadi, Mehdi, Mehran Javanmardi, and Tolga Tasdizen (2016). "Regularization with stochastic transformations and perturbations for deep semi-supervised learning". In: *Advances in neural information processing systems* 29, pp. 1163–1171.

Salehi, Mohammadreza et al. (2021). "A unified survey on anomaly, novelty, open-set, and out-of-distribution detection: Solutions and future challenges". In: *arXiv preprint arXiv:2110.14051*.

Samek, Wojciech, Thomas Wiegand, and Klaus-Robert Müller (2017). "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models". In: *arXiv preprint arXiv:1708.08296*.

Sanh, Victor et al. (2019). "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *arXiv preprint arXiv:1910.01108*.

Saravia, Elvis (Dec. 2022). "Prompt Engineering Guide". In: <https://github.com/dair-ai/Prompt-Engineering-Guide>.

Särndal, Carl-Erik, Bengt Swensson, and Jan Wretman (2003). *Model assisted survey sampling*. Springer Science & Business Media.

Schlegl, Thomas et al. (2017). "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery". In: *International conference on information processing in medical imaging*. Springer, pp. 146–157.

Schlichtkrull, Michael et al. (2018). "Modeling relational data with graph convolutional networks". In: *European semantic web conference*. Springer, pp. 593–607.

Schmidhuber, Jürgen (2015). "Deep learning in neural networks: An overview". In: *Neural networks* 61, pp. 85–117.

Schölkopf, Bernhard et al. (2001). "Estimating the support of a high-dimensional distribution". In: *Neural computation* 13.7, pp. 1443–1471.

Schubert, Erich et al. (2017). "DBSCAN revisited, revisited: why and how you should (still) use DBSCAN". In: *ACM Transactions on Database Systems (TODS)* 42.3, pp. 1–21.

Schuster, Peter (2000). "Taming combinatorial explosion". In: *Proceedings of the National Academy of Sciences* 97.14, pp. 7678–7680.

Seliya, Naeem, Azadeh Abdollah Zadeh, and Taghi M Khoshgoftaar (2021). "A literature review on one-class classification and its potential applications in big data". In: *Journal of Big Data* 8.1, pp. 1–31.

Sellam, Thibault, Dipanjan Das, and Ankur P Parikh (2020). "BLEURT: Learning robust metrics for text generation". In: *arXiv preprint arXiv:2004.04696*.

Sener, Ozan and Vladlen Koltun (2018). "Multi-task learning as multi-objective optimization". In: *arXiv preprint arXiv:1810.04650*.

Shah, Sohil Atul and Vladlen Koltun (2017). "Robust continuous clustering". In: *Proceedings of the National Academy of Sciences* 114.37, pp. 9814–9819.

— (2018). "Deep continuous clustering". In: *arXiv preprint arXiv:1803.01449*.

Shao, Zhou et al. (2021). "Evolutions and trends of artificial intelligence (AI): research, output, influence and competition". In: *Library Hi Tech*.

Sharma, Sagar and Simone Sharma (2017). "Activation functions in neural networks". In: *Towards Data Science* 6.12, pp. 310–316.

Shaw, Peter, Jakob Uszkoreit, and Ashish Vaswani (2018). "Self-attention with relative position representations". In: *arXiv preprint arXiv:1803.02155*.

Shen, Li et al. (2023). "On Efficient Training of Large-Scale Deep Learning Models: A Literature Review". In: *arXiv preprint arXiv:2304.03589*.

Shevlyakov, Georgy, Stephan Morgenthaler, and Alexander Shurygin (2008). "Re-descending M-estimators". In: *Journal of Statistical Planning and Inference* 138.10, pp. 2906–2917.

Shin, Taylor et al. (2020). "Autoprompt: Eliciting knowledge from language models with automatically generated prompts". In: *arXiv preprint arXiv:2010.15980*.

Sibai, Rayane El et al. (2016). "Sampling algorithms in data stream environments". In: *2016 International Conference on Digital Economy (ICDEC)*, pp. 29–36. DOI: 10.1109/ICDEC.2016.7563142.

Sidorov, Oleksii et al. (2020). "Textcaps: a dataset for image captioning with reading comprehension". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II* 16. Springer, pp. 742–758.

Singh, Dalwinder and Birmohan Singh (2020). "Investigating the impact of data normalization on classification performance". In: *Applied Soft Computing* 97, p. 105524.

Singh, Karanjit and Shuchita Upadhyaya (2012). "Outlier detection: applications and techniques". In: *International Journal of Computer Science Issues (IJCSI)* 9.1, p. 307.

Singh, Yogendra Narain and Phalguni Gupta (2007). "Quantitative evaluation of normalization techniques of matching scores in multimodal biometric systems". In: *International Conference on Biometrics*. Springer, pp. 574–583.

Sloman, Aaron (1971). "Interactions between philosophy and artificial intelligence: The role of intuition and non-logical reasoning in intelligence". In: *Artificial intelligence* 2.3-4, pp. 209–225.

Smith, Helen (2021). "Clinical AI: opacity, accountability, responsibility and liability". In: *AI SOCIETY* 36, pp. 535–545. DOI: 10.1007/s00146-020-01019-6.

Smiti, Abir (2020). "A critical overview of outlier detection methods". In: *Computer Science Review* 38, p. 100306.

Smola, Alex et al. (2007). "A Hilbert space embedding for distributions". In: *International Conference on Algorithmic Learning Theory*. Springer, pp. 13–31.

Sobol, Ilya M (2001). "Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates". In: *Mathematics and computers in simulation* 55.1-3, pp. 271–280.

Sobolevsky, Stanislav (2021). "Hierarchical graph neural networks". In: *arXiv preprint arXiv:2105.03388*.

Song, Le et al. (2009). "Hilbert space embeddings of conditional distributions with applications to dynamical systems". In: *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 961–968.

Souvenir, Richard and Robert Pless (2005). "Manifold clustering". In: *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*. Vol. 1. IEEE, pp. 648–653.

Sra, Suvrit, Sebastian Nowozin, and Stephen J Wright (2012). *Optimization for machine learning*. Mit Press.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1, pp. 1929–1958.

Su, Han et al. (2020). "A survey of trajectory distance measures and performance evaluation". In: *The VLDB Journal* 29, pp. 3–32.

Subasi, Abdulhamit (2020). "Chapter 2 - Data preprocessing". In: *Practical Machine Learning for Data Analysis Using Python*. Ed. by Abdulhamit Subasi. Academic Press, pp. 27–89. ISBN: 978-0-12-821379-7. DOI: <https://doi.org/10.1016/B978-0-12-821379-7.00002-3>. URL: <https://www.sciencedirect.com/science/article/pii/B9780128213797000023>.

Sun, Shiliang et al. (2019). "A survey of optimization methods from a machine learning perspective". In: *IEEE transactions on cybernetics* 50.8, pp. 3668–3681.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Suzuki, Kenji (2013). *Artificial neural networks: Architectures and applications*. BoD—Books on Demand.

Swirszcz, Grzegorz, Wojciech Marian Czarnecki, and Razvan Pascanu (2016). "Local minima in training of neural networks". In: *arXiv preprint arXiv:1611.06310*.

Sze, Vivienne et al. (2017). "Hardware for machine learning: Challenges and opportunities". In: *2017 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, pp. 1–8.

Szegedy, Christian et al. (2015). "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Szegedy, Christian et al. (2016). "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826.

Szegedy, Christian et al. (2017). "Inception-v4, inception-resnet and the impact of residual connections on learning". In: *Thirty-first AAAI conference on artificial intelligence*.

Tan, Xiaoyang and Bill Triggs (2010). "Enhanced local texture feature sets for face recognition under difficult lighting conditions". In: *IEEE transactions on image processing* 19.6, pp. 1635–1650.

Tao, Yaguang et al. (2021). "A comparative analysis of trajectory similarity measures". In: *GIScience & Remote Sensing* 58.5, pp. 643–669.

Tarvainen, Antti and Harri Valpola (2017). "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results". In: *arXiv preprint arXiv:1703.01780*.

Tay, Yi et al. (2020). "Efficient transformers: A survey". In: *arXiv preprint arXiv:2009.06732*.

Teboul, A. (2023). *Diabetes Health Indicators Dataset*. <https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset>.

Tenenbaum, Joshua B, Vin De Silva, and John C Langford (2000). "A global geometric framework for nonlinear dimensionality reduction". In: *science* 290.5500, pp. 2319–2323.

Thomas, Georg B (1992). "Calculus and analytic geometry". In: *Massachusetts Institute of Technology, Massachusetts, USA, Addison-Wesley Publishing Company, ISBN: 0-201-60700-X*.

Thompson, Neil C et al. (2020). "The computational limits of deep learning". In: *arXiv preprint arXiv:2007.05558*.

Thrun, Sebastian B (1992). "Efficient exploration in reinforcement learning". In.

Tian, Jun (2003). "Reversible data embedding using a difference expansion". In: *IEEE transactions on circuits and systems for video technology* 13.8, pp. 890–896.

Tieleman, Tijmen, Geoffrey Hinton, et al. (2012). "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* 4.2, pp. 26–31.

Tiwari, Ashish (2022). "Chapter 2 - Supervised learning: From theory to applications". In: *Artificial Intelligence and Machine Learning for EDGE Computing*. Ed. by Rajiv Pandey et al. Academic Press, pp. 23–32. ISBN: 978-0-12-824054-0. DOI: <https://doi.org/10.1016/B978-0-12-824054-0.00026-5>. URL: <https://www.sciencedirect.com/science/article/pii/B9780128240540000265>.

Tomašev, Nenad and Miloš Radovanović (2016). "Clustering evaluation in high-dimensional data". In: *Unsupervised learning algorithms*. Springer, pp. 71–107.

Toohey, Kevin and Matt Duckham (2015). "Trajectory similarity measures". In: *Sigspatial Special* 7.1, pp. 43–50.

Touvron, Hugo et al. (2023). "Llama 2: Open foundation and fine-tuned chat models". In: *arXiv preprint arXiv:2307.09288*.

Trabelsi, Chiheb et al. (2017). "Deep complex networks". In: *arXiv preprint arXiv:1705.09792*.

Tricomi, Francesco Giacomo, Arthur Erdélyi, et al. (1951). "The asymptotic expansion of a ratio of gamma functions". In: *Pacific J. Math* 1.1, pp. 133–142.

Turc, Iulia et al. (2019). "Well-read students learn better: On the importance of pre-training compact models". In: *arXiv preprint arXiv:1908.08962*.

Vaicenavicius, Juozas et al. (2019). "Evaluating model calibration in classification". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 3459–3467.

Valpola, Harri (2015). "From neural PCA to deep unsupervised learning". In: *Advances in independent component analysis and learning machines*. Elsevier, pp. 143–171.

Van Der Maaten, Laurens, Eric Postma, Jaap Van den Herik, et al. (2009). "Dimensionality reduction: a comparative". In: *J Mach Learn Res* 10.66-71, p. 13.

Van Dyk, David A and Xiao-Li Meng (2001). "The art of data augmentation". In: *Journal of Computational and Graphical Statistics* 10.1, pp. 1–50.

Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in neural information processing systems*, pp. 5998–6008.

Veličković, Petar et al. (2017). "Graph attention networks". In: *arXiv preprint arXiv:1710.10903*.

Vellido, Alfredo, Paulo JG Lisboa, and J Vaughan (1999). "Neural networks in business: a survey of applications (1992–1998)". In: *Expert Systems with applications* 17.1, pp. 51–70.

Villalobos, Pablo et al. (2022). "Machine learning model sizes and the parameter gap". In: *arXiv preprint arXiv:2207.02852*.

Vincent, Pascal et al. (2010). "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." In: *Journal of machine learning research* 11.12.

Vitter, Jeffrey S (1985). "Random sampling with a reservoir". In: *ACM Transactions on Mathematical Software (TOMS)* 11.1, pp. 37–57.

Von Luxburg, Ulrike (2007). "A tutorial on spectral clustering". In: *Statistics and computing* 17, pp. 395–416.

Wan, Li et al. (2013). "Regularization of neural networks using dropconnect". In: *International conference on machine learning*. PMLR, pp. 1058–1066.

Wang, Biao et al. (2023). "Selective Feature Bagging of one-class classifiers for novelty detection in high-dimensional data". In: *Engineering Applications of Artificial Intelligence* 120, p. 105825.

Wang, Fei-Yue et al. (2016a). "Where does AlphaGo go: From church-turing thesis to AlphaGo thesis and beyond". In: *IEEE/CAA Journal of Automatica Sinica* 3.2, pp. 113–120.

Wang, Haozhou et al. (2013). "An effectiveness study on trajectory similarity measures". In: *Proceedings of the Twenty-Fourth Australasian Database Conference-Volume 137*, pp. 13–22.

Wang, Hongzhi, Mohamed Jaward Bah, and Mohamed Hammad (2019). "Progress in outlier detection techniques: A survey". In: *Ieee Access* 7, pp. 107964–108000.

Wang, Sinong et al. (2020a). "Linformer: Self-attention with linear complexity". In: *arXiv preprint arXiv:2006.04768*.

Wang, Wei et al. (2016b). "Database meets deep learning: Challenges and opportunities". In: *ACM SIGMOD Record* 45.2, pp. 17–22.

Wang, Xiao et al. (2022a). "A survey on heterogeneous graph embedding: methods, techniques, applications and sources". In: *IEEE Transactions on Big Data*.

Wang, Xuezhi et al. (2022b). "Self-consistency improves chain of thought reasoning in language models". In: *arXiv preprint arXiv:2203.11171*.

Wang, Yi et al. (2020b). "Novelty detection and online learning for chunk data streams". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.7, pp. 2400–2412.

Wang, Ziyu et al. (2016c). "Dueling network architectures for deep reinforcement learning". In: *International conference on machine learning*. PMLR, pp. 1995–2003.

Wegmann, Marc et al. (2021). "A review of systematic selection of clustering algorithms and their evaluation". In: *arXiv preprint arXiv:2106.12792*.

Wei, Jason et al. (2022). "Chain-of-thought prompting elicits reasoning in large language models". In: *Advances in Neural Information Processing Systems* 35, pp. 24824–24837.

Weidinger, Laura et al. (2021). "Ethical and social risks of harm from language models". In: *arXiv preprint arXiv:2112.04359*.

Weisstein, Eric W. (n.d.). "Sample Variance Computation." In: *MathWorld—A Wolfram Web Resource* (). URL: <https://mathworld.wolfram.com/SampleVarianceComputation.html>.

Weiszfeld, Endre (1936). "Sur un problème de minimum dans l'espace". In: *Tohoku Mathematical Journal, First Series* 42, pp. 274–280.

Welford, BP (1962). "Note on a method for calculating corrected sums of squares and products". In: *Technometrics* 4.3, pp. 419–420.

Weng, Lilian (2017). "An Overview of Deep Learning for Curious People". In: *lilianweng.github.io/lil-log*. URL: <https://lilianweng.github.io/lil-log/2017/06/21/an-overview-of-deep-learning.html>.

— (2019). "Self-Supervised Representation Learning". In: *lilianweng.github.io/lil-log*. URL: <https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html>.

— (2021). "Learning with not Enough Data Part 1: Semi-Supervised Learning". In: *lilianweng.github.io*. URL: <https://lilianweng.github.io/posts/2021-12-05-semi-supervised/>.

— (2023). "Prompt Engineering". In: *lilianweng.github.io*. URL: <https://lilianweng.github.io/posts/2023-03-15-prompt-engineering/>.

Wenzek, Guillaume et al. (2019). "CCNet: Extracting high quality monolingual datasets from web crawl data". In: *arXiv preprint arXiv:1911.00359*.

Winston., Patrick (2010). "MIT OpenCourseWare 6.034 Artificial Intelligence". In.

Wolf, Thomas et al. (2020a). "Transformers: State-of-the-art natural language processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45.

Wolf, Thomas et al. (Oct. 2020b). "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, pp. 38–45. URL: <https://www.aclweb.org/anthology/2020.emnlp-demos.6>.

Wong, Sebastien C et al. (2016). "Understanding data augmentation for classification: when to warp?" In: *2016 international conference on digital image computing: techniques and applications (DICTA)*. IEEE, pp. 1–6.

Wu, Xindong et al. (2013). "Data mining with big data". In: *IEEE transactions on knowledge and data engineering* 26.1, pp. 97–107.

Wu, Zonghan et al. (2020). "A comprehensive survey on graph neural networks". In: *IEEE transactions on neural networks and learning systems* 32.1, pp. 4–24.

Xiao, Guangxuan et al. (2023). "Smoothquant: Accurate and efficient post-training quantization for large language models". In: *International Conference on Machine Learning*. PMLR, pp. 38087–38099.

Xiao, Qinfeng et al. (2021). "Unsupervised anomaly detection with distilled teacher-student network ensemble". In: *Entropy* 23.2, p. 201.

Xu, Dongkuan and Yingjie Tian (2015). "A comprehensive survey of clustering algorithms". In: *Annals of Data Science* 2.2, pp. 165–193.

Xu, Feiyu et al. (2019). "Explainable AI: A brief survey on history, research areas, approaches and challenges". In: *Natural Language Processing and Chinese Computing: 8th CCF International Conference, NLPCC 2019, Dunhuang, China, October 9–14, 2019, Proceedings, Part II 8*. Springer, pp. 563–574.

Xu, Keyulu et al. (2018). "How powerful are graph neural networks?" In: *arXiv preprint arXiv:1810.00826*.

Xu, Rui and Don Wunsch (2008). *Clustering*. Vol. 10. John Wiley & Sons.

Xu, Rui and Donald Wunsch (2005). "Survey of clustering algorithms". In: *IEEE Transactions on neural networks* 16.3, pp. 645–678.

Yang, Jingkang et al. (2022). "OpenOOD: Benchmarking generalized out-of-distribution detection". In: *arXiv preprint arXiv:2210.07242*.

Yang, Peilun et al. (2021). "T3s: Effective representation learning for trajectory similarity computation". In: *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, pp. 2183–2188.

Yazan, Ersan and M Fatih Talu (2017). "Comparison of the stochastic gradient descent based optimization techniques". In: *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*. IEEE, pp. 1–5.

Ye, Seonghyeon et al. (2023). "In-context instruction learning". In: *arXiv preprint arXiv:2302.14691*.

Ying, Xue (2019). "An overview of overfitting and its solutions". In: *Journal of physics: Conference series*. Vol. 1168. IOP Publishing, p. 022022.

Yu, Yaodong et al. (2020). "Learning diverse and discriminative representations via the principle of maximal coding rate reduction". In: *Advances in Neural Information Processing Systems* 33, pp. 9422–9434.

Yuan, Weizhe, Graham Neubig, and Pengfei Liu (2021). "Bartscore: Evaluating generated text as text generation". In: *Advances in Neural Information Processing Systems* 34, pp. 27263–27277.

Yun, Seongjun et al. (2019). "Graph transformer networks". In: *Advances in neural information processing systems* 32.

Zadrozny, Bianca and Charles Elkan (2001). "Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers". In: *Icml*. Vol. 1, pp. 609–616.

— (2002). "Transforming classifier scores into accurate multiclass probability estimates". In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 694–699.

Zagoruyko, Sergey and Nikos Komodakis (2016). "Wide residual networks". In: *arXiv preprint arXiv:1605.07146*.

Zbontar, Jure et al. (2021). "Barlow twins: Self-supervised learning via redundancy reduction". In: *arXiv preprint arXiv:2103.03230*.

Zhang, Chuxu et al. (2019). "Heterogeneous graph neural network". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 793–803.

Zhang, Daokun et al. (2018). "Network representation learning: A survey". In: *IEEE transactions on Big Data* 6.1, pp. 3–28.

Zhang, Guoqiang Peter (2000). "Neural networks for classification: a survey". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 30.4, pp. 451–462.

Zhang, Richard, Phillip Isola, and Alexei A Efros (2017). "Split-brain autoencoders: Unsupervised learning by cross-channel prediction". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1058–1067.

Zhang\*, Tianyi et al. (2020). "BERTScore: Evaluating Text Generation with BERT". In: *International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=SkeHuCVFD>.

Zhang, Ziming and Venkatesh Saligrama (2015). "Zero-shot learning via semantic similarity embedding". In: *Proceedings of the IEEE international conference on computer vision*, pp. 4166–4174.

Zhao, Yue, Zain Nasrullah, and Zheng Li (2019). "PyOD: A Python Toolbox for Scalable Outlier Detection". In: *Journal of Machine Learning Research* 20.96, pp. 1–7. URL: <http://jmlr.org/papers/v20/19-011.html>.

Zhao, Yue et al. (2021). "Suod: Accelerating large-scale unsupervised heterogeneous outlier detection". In: *Proceedings of Machine Learning and Systems* 3, pp. 463–478.

Zhong, Zhun et al. (2020). "Random erasing data augmentation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 07, pp. 13001–13008.

Zhou, Lina et al. (2017). "Machine learning on big data: Opportunities and challenges". In: *Neurocomputing* 237, pp. 350–361.

Zhou, Sheng et al. (2022a). "A comprehensive survey on deep clustering: Taxonomy, challenges, and future directions". In: *arXiv preprint arXiv:2206.07579*.

Zhou, Yongchao et al. (2022b). "Large language models are human-level prompt engineers". In: *arXiv preprint arXiv:2211.01910*.

Zhou, Zhi-Hua (2018). "A brief introduction to weakly supervised learning". In: *National science review* 5.1, pp. 44–53.

Zhu, Shichao et al. (2019). "Relation structure-aware heterogeneous graph neural network". In: *2019 IEEE international conference on data mining (ICDM)*. IEEE, pp. 1534–1539.

Zhu, Xiaojin and Andrew B Goldberg (2009). "Introduction to semi-supervised learning". In: *Synthesis lectures on artificial intelligence and machine learning* 3.1, pp. 1–130.

Zhu, Xiaojin Jerry (2005). *Semi-supervised learning literature survey*. University of Wisconsin-Madison Department of Computer Sciences.

Zhuang, Fuzhen et al. (2020). "A Comprehensive Survey on Transfer Learning". In: *arXiv preprint arXiv:1911.02685*.