Assessment and Analysis of the Applicability of Recurrent Neural Networks to Natural Language Understanding with a Focus on the Problem of Coreference Resolution

Inaugural-Dissertation
zur Erlangung des Doktorgrades der Philosophie
an der Ludwig–Maximilians–Universität
München

vorgelegt von
Franz David Kaumanns
aus
München

2016

Erstgutachter: Prof. Dr. Hinrich Schütze
Zweitgutachter: Dr. Helmut Schmid
Datum der mündlichen Prüfung: 06.07.2016

Contents

Zusammenfassung **vii**

Abstract **ix**

Acknowledgments **xi**

1 Introduction **1**
1.1 Contributions 1
1.2 Structure 3

2 Toward A Connectionist Understanding Of Language **5**
2.1 Parallel Distributed Cognition 5
2.2 Connectionism vs Classicist Model of Cognition 6
2.3 Connectionist Models of Grammar Learning 7
2.4 Issues of Modern Applications of Connectionism for NLP 8
2.5 Scaling Down: Toy Tasks of Natural Language Understanding 10

3 Recurrent Neural Networks **13**
3.1 The Recurrent Framework 13
3.1.1 Introduction 14
3.1.2 Limitations and Solutions 15
3.1.3 Recurrent Language Processing 16
3.2 Long Short-Term Memory 17
3.2.1 Architecture 19
3.2.2 Alternatives 22
3.2.3 Algorithm 23
3.3 Training Recurrent Neural Networks 28
3.3.1 Error and Backpropagation 28
3.3.2 Stochastic Gradient Descent and Learning oRate Decay 29
3.3.3 Parameter-Wise Learning Rate Adaptation 30
3.4 Bidirectional Recurrency 31

3.4.1 Algorithm . . . 32

4 Coreference Resolution **35**
4.1 Introduction . . . 35
4.1.1 Mention-Pair Model . . . 36
4.1.2 Understanding Language by Resolving Coreferences . . . 37
4.2 Features of Coreferences . . . 38
4.3 Rule-Based Systems of Coreference Resolution . . . 39
4.4 Overview of Machine Learning Approaches to Coreference Resolution . . . 40
4.4.1 Supervised Models of Coreference Resolution . . . 40
4.4.2 Unsupervised Approaches . . . 41

5 Predictive Anaphora Resolution **43**
5.1 Related Work . . . 44
5.2 Casting Anaphora Resolution to Sequence Classification . . . 44
5.2.1 Meaning Function . . . 45
5.2.2 Data Design . . . 46
5.3 Synthesizing Data . . . 46
5.3.1 Generative Grammar . . . 47
5.3.2 Features . . . 48
5.4 Setup . . . 48
5.4.1 Architecture . . . 49
5.4.2 Error and Backpropagation . . . 50
5.4.3 Optimization . . . 50
5.4.4 Accuracy and Baseline . . . 51
5.4.5 Set Ratios . . . 51
5.5 Evaluation . . . 51
5.6 Neural Analysis . . . 58
5.6.1 Related Work . . . 58
5.6.2 States of Gates . . . 59
5.6.3 Heatmaps . . . 61
5.7 Conclusion . . . 78

6 Mention Pair Identification **81**
6.1 Related Work . . . 82
6.2 Data Preparation . . . 83
6.2.1 Overview . . . 83
6.2.2 Reformat CoNLL-2012 . . . 85
6.2.3 Normalization . . . 86
6.2.4 Splitting CoNLL-2012 Into Sentences . . . 87
6.2.5 Creating Powersets of Mention Pairs . . . 87

6.2.6 Data Reduction . . . 89
6.2.7 Merging and Shuffling Samples . . . 90
6.2.8 Masking Rare Tokens . . . 90
6.2.9 Creating the Data Sets . . . 90
6.3 Setup . . . 91
6.3.1 Architectures . . . 91
6.3.2 Parameters . . . 92
6.3.3 Optimization . . . 92
6.4 Evaluation . . . 94
6.5 Linguistic Analysis . . . 97
6.5.1 Definitions . . . 98
6.5.2 Examples for Mention Classes . . . 99
6.5.3 Evaluation of Class Performances . . . 100
6.6 Conclusion . . . 106

7 Visual Analysis of Recurrent Neural Networks 109
7.1 Related Work . . . 110
7.1.1 Performance Analysis . . . 110
7.1.2 Low-Level Analysis of Image Classifiers . . . 111
7.1.3 Low-Level Analysis of Language Models . . . 112
7.2 Methodology . . . 113
7.2.1 Terminology . . . 113
7.2.2 Framework . . . 116
7.3 Gate Saturations . . . 118
7.3.1 Differences Between CharRNN and SCRNN . . . 118
7.3.2 Definition of Gate Saturations . . . 119
7.3.3 Hypothesis . . . 120
7.3.4 Saturations at All Time Steps . . . 121
7.3.5 Saturations at Mention Markers . . . 122
7.3.6 Saturations at Query Tokens . . . 124
7.3.7 Summary . . . 125
7.4 States of Memory Cells . . . 128
7.4.1 Dimensionality Reduction via t-SNE . . . 128
7.4.2 Filtered vs Unfiltered Data . . . 129
7.4.3 Cell States at All Time Steps . . . 129
7.4.4 Cell States at Anchor Tokens . . . 129
7.5 Conclusion . . . 135

8 Conclusion 139
8.1 Summary . . . 139
8.2 Future Work . . . 140

9 Appendix **143**
9.1 Prolog Code for Generating Synthetic Corpus 143
9.2 Example Sentences from CoNLL-2012 146
9.2.1 True Positive Evaluation Set 146
9.2.2 True Negative Evaluation Set 147
9.2.3 False Positive Evaluation Set 147
9.2.4 False Negative Evaluation Set 148
9.3 JSON Specification for DeepDiver 150

List of Figures **151**

List of Tables **157**

References **159**

Zusammenfassung

Recurrent Neural Networks erweitern das konnektionistische Prinzip der Informationsverarbeitung um eine grundlegende Fähigkeit biologischer Kognition: temporale Modellierung sequentieller Eingaben. Diese Erweiterung erlaubt einem künstlichen neuronalen Netzwerk, grammatische Muster aus natürlichen Sprachdaten zu lernen und auf eine Aufgabe anzuwenden.

Ein lähmendes Problem des Trainings rekurrenter Netzwerke, das "vanishing gradient"-Problem, führte in den neunziger Jahren zur Entwicklung neuer Optimierungsverfahren und Speicherkontrollmechanismen zweiter Ordnung, wie Long Short-Term Memory (LSTM), deren Erfolg jedoch zunächst von den höheren Rechenanforderungen gedämpft wurde. Mit der wachsenden Verfügbarkeit von Rechenleistung und Daten verschob sich der Forschungsfokus bald von der Grundlagenforschung zu Performanzanalysen verschiedener Netzwerkkonfigurationen. Obwohl die Analyse neuronaler Netzwerkmodelle über die Jahre vereinzelt weitergeführt wurde, schien die Verarbeitung natürlicher Sprache weitestgehend außen vor zu bleiben.

Diese Arbeit trägt zu kürzlichem und historischem Bestreben bei, die Verarbeitung von natürlicher Sprache von der Eingabeebene bis zur Entscheidungsebene zu verfolgen. Wir untersuchen, wie sich moderne tiefschichtige Netzwerktechnologien einsetzen lassen, um einem wichtigen Ziel der Konnektionismus-Forschung näher zu kommen: das Verständnis über die Verarbeitung von sprachlicher Bedeutung in einem komplexen Netzwerk künstlicher Neuronen.

Hierzu verfolgen wir einen Ansatz, der die Komplexität der Experimente iterativ erhöht. Korpora, Algorithmen und Aufgaben werden zunächst kontrolliert reduziert und vereinfacht. Rekurrente LSTM-Klassifizierer werden auf Wortsequenzen aus künstlichen Daten sowie simplen natürlichen Daten trainiert, um ein wohldefiniertes aber schwieriges Problem der Sprachverarbeitung zu lösen: Anapher-Resolution und paarweise Koreferenzresolution. Linguistische und visuelle Analysemethoden erlauben einen intuitiven Einblick in die Aktivität künstlicher Neuronen.

Wir hoffen, dass diese Arbeit einen Beitrag leistet für einen wissenschaftlichen Paradigmawandel in Richtung eines tieferen Verständnisses über das komplexe Uhrwerk künst-

licher neuronaler Netzwerke.

Abstract

Recurrent Neural Networks extend the connectionist paradigm of information processing with a crucial ability of biological cognition: temporal modeling of sequential input. This extension allows an Artificial Neural Network to learn and apply grammatical patterns from natural language data.

In the 1990s, a crippling issue in training recurrent networks, the vanishing gradient problem, was tackled by new optimization techniques and second-order gating mechanisms, such as Long Short-Term Memory (LSTM) units, though their successes were initially dampened by their even higher computational complexity. As technology gradually met the heavy requirements of Deep Learning and vast amounts of training data became available, the focus of research shifted from its initial low-level scientific analysis to high-level performance evaluations. Despite an abundance of work on inspection of deep networks for image processing tasks, neural models for Natural Language Processing have so far largely remained outside the scope of low-level analysis. Deep Learning of natural language has become a black box.

This work contributes to recent and historic efforts to understand how natural language input is transformed by a neural network into high-level decisions. We investigate the utilization of latest technologies and insights about Deep Learning in order to come a little closer to a future milestone of connectionist research: understanding how natural language meaning is processed in a complex network of artificial neurons.

In order to reach this goal, we pursue a bottom-up approach to experimental complexity. Corpora, algorithms, and task difficulty are scaled down in a controlled fashion. Recurrent LSTM classifiers are trained on token sequences from both artificial and restricted real-life data to solve a well-defined but difficult subtask of natural language understanding: Anaphora Resolution and identification of coreferent mention pairs. Various methods of linguistic and visual analysis help gain insights into neural activity across samples, layers, and time.

We hope that this work will prove to be part of a greater scientific shift toward understanding what actually makes a Deep Neural Network tick.

Acknowledgments

This thesis is the result of countless discussions with my friends and colleagues without whom I cannot imagine this work to have finished.

Above all, Prof Dr Hinrich Schütze has provided me with the chance to explore an exciting and world-changing field of research. I am deeply grateful for his time and guidance that helped me learn and accumulate insights as never before. The education which I have had the privilege to gain under his supervision has been and will be the groundwork for my future endeavours.

Dr Helmut Schmid gave me the privilege of his time, valuable advice, suggestions, and unobfuscated criticism for my thesis. For this and for our revealing discussions around the algorithmic and mathematical details of my work I want to express my sincere appreciation.

I owe my friend and colleague Dr Uwe Springmann the most profound gratitude for his opinions and experienced advice, and for sharing his excitement and curiosity for the science of things. Conversely, and yet completely similarly, my friend and brother-in-arms Daniel Bruder helped me gain both deeper and higher insights about the engineering of things, and the things of matter. I owe special thanks to my friend and mentor Dr Maximilian Hadersbeck who has guided and inspired me through almost a decade of studies at CIS. The same holds for my friend Dr Hans Leiß who has never grown tired of holding up the flag of uncompromising teaching and research in what has made linguistics a true window to our cognition. If there is one belief that binds our sworn group of investigators and tweakers, then it is the conviction that passion and child-like wonder are a rewarding combination, both professionally and personally.

My most grateful thanks go my friends and colleagues, Sebastian Ebert, Yadollah Yaghoobzadeh, Irina Sergienya, and Dr Desislava Zhekova with whom I had more discussions and inspirational exchanges than I can remember. Their invaluable ideas fueled my work and paved my path.

Last but not least, I had the privilege of gaining indispensable practical experience during my internships with the most encouraging, creative, and brightest team of engineers I can

imagine. My special thanks go to Dr Hans Dolfing, Dr Jerome Bellegarda, and Dr Sibel Yaman from whom I learned both theory and praxis, and who helped me kickstart the ideas that lead to this thesis.

Chapter 1

Introduction

1.1 Contributions

The goal of this thesis is to advance efforts for a low-level analysis of Deep Neural Networks for Natural Language Understanding (NLU). We strive for a continuation of the original work on Connectionist models in the 1980s and 1990s that were driven by the desire for a basic understanding of their neural dynamics.

Contemporary Artificial Neural Network research community is facing the same threat to its scientific methodology that the theoretical physicist Richard Feynman was already shocked by with his colleagues at the National Accelerator Laboratory (NAL), as stated in his 1974 commencement address at Caltech (Feynman, 1998):

> *And so the men in charge of programs at NAL are so anxious for new results, in order to get more money to keep the thing going for public relations purposes, they are destroying – possibly – the value of the experiments themselves, which is the whole purpose of the thing. It is often hard for the experimenters there to complete their work as their scientific integrity demands.*

Similar concerns have been expressed just recently by various figures in the field of Natural Language Processing (see Section 2.4). In the past decades, research efforts especially around the new trend surrounding Artificial Neural Networks and Deep Learning has steadily shifted toward high performance comparisons. This is not the least due to a substantial financial support from the industry that is eager to fund research programs and draw scientists into their result-oriented realm of real-life applications. In a recent interview, Yann LeCun, one of the driving forces behind the revival of convolutional networks, emphasized the idea that scientific investigation entails a profound analysis

and understanding, not just engineering and application.[1]

This thesis is an effort to take a step back from these powerful yet black-boxed systems, and instead join the research forces that try to relate high-level model decisions back to the triggering inputs. Specifically, we take a look under the hood of Recurrent Neural Networks (RNN) and watch them solve tasks that require complex reasoning and abstraction over the linguistic input. We show that RNNs, though providing an indispensable time dimension for NLP tasks, require a more sophisticated framework of linguistic analysis in order to trace high-level predictions back to individual inputs and neurons across several time steps. We therefore pursue a bottom-up approach with respect to the complexity of the following components:

Architecture

We show that the gating mechanisms provided by Long Short-Term Memory (LSTM) networks exhibit interesting dynamics and help understand the roles of single memory cells that would be obfuscated by the memory compression employed in simple RNNs. Combined with a bidirectional mode of training, this architecture reveals insightful patterns of input processing.

Task

Variants of this architecture are applied to the tasks of Coreference Resolution (CR) and Anaphora Resolution (AR), both of which require complex reasoning over natural language input when performed on real-life data. We show that these well-defined problems help us downscale the complexity of the challenge of Natural Language Understanding (NLU). Furthermore, they can easily be cast to problems of sequence classification and are thus a perfect fit for RNNs.

Data

Keeping in mind our goal of intuitive inspection of the resulting models, we propose two methods for designing the data:

1. Toy corpus with restricted linguistic complexity
2. Reduced real-life natural language corpus

We will show that the particular sample layouts of these data sets allow us to not only look out for certain expected features, but also to probe the model at particular time steps along which all samples can be aligned.

Based on these components, we apply four methods of low-level analysis:

- Heatmaps over single neurons of tiny networks

[1] http://www.sueddeutsche.de/digital/kuenstliche-intelligenz-wie-google-und-facebook-computern-denken-beibringen-1.2885204-3

- Evaluation of sample subsets corresponding to linguistic phenomena
- Scatter plots of gate activations across time
- t-SNE analysis of memory cell states across time

The evaluation of the resulting visualizations provide interesting insights about the low-level dynamics of our models.

The *Torch* code for reproducing the results in this thesis is published as follows:

- **DeepKaspar** (https://github.com/kaumanns/DeepKaspar) – Tools for training and logging binary and non-binary sequence classifiers with Recurrent Long Short-Term Memory.
- **DeepDiver** (https://github.com/kaumanns/DeepDiver) – Tools for creating graphs, scatter plots, and heatmaps from JSON-formatted neural network logs.

1.2 Structure

The chapters of this thesis are structured as follows:

Section 2 outlines the theoretical foundation of Artificial Neural Networks. We discuss core ideas, motivations, and salient applications of the connectionist paradigm in the field of Natural Language Processing (NLP) and Natural Language Understanding (NLU).

Section 3 describes the principles and the architecture of Recurrent Neural Networks (RNN), including important extensions to the original recurrent framework: Long Short-Term Memory (LSTM) units and bidirectionality. Algorithms and equations for the training of LSTM networks are presented in detail.

Section 4 provides an overview of the field of Coreference Resolution (CR) and Anaphora Resolution (AR), two well-defined NLP tasks that require higher-level reasoning over the language input. We summarize their linguistic principles and introduce the core ideas and examples of both rule-based approaches and machine learning models.

Section 5 reframes the task of AR as a problem of sequence classification with a recurrent LSTM network. It describes how a toy data set of synthetic sentences can boil down the problem of natural language understanding analysis to a feasible task. The chapter concludes with a first visual analysis of single neural activations.

Section 6 extends the previous chapter with the binary classification of mention-pairs in a real-life natural language data set. It describes how the challenge data from CoNLL-2012 is reshaped and reduced in order to facilitate sequence classification with an LSTM network. The chapter concludes with a performance analysis on specific test data subsets corresponding to different linguistic phenomena.

Section 7 applies visualizations of different layer subsets of the states of a bidirectional LSTM network. It uncovers interesting dynamics within the memory cells and gates along fixed anchor tokens within the sample sequences.

Section 8 concludes with a summary of the insights gained and with directions for further analysis and future work.

Chapter 2

Toward A Connectionist Understanding Of Language

This chapter summarizes the rise and early advancements of *Parallel Distributed Processing*, an implementation of the cognitive paradigm of *connectionism* that inspired the development of Deep Neural Networks (DNN).

Section 2.1 outlines the motivation and principles of connectionism.

Section 2.2 provides an overview of the contrast between connectionist models and classicist views of cognition.

Section 2.3 explains how connectionism motivated early recurrent networks and their application to natural language tasks.

Section 2.4 discusses contemporary issues of the connectionist approach to solving NLP tasks.

Section 2.5 outlines the history and motivation for a controlled downscaling of the data complexity in order to make low-level analysis feasible.

2.1 Parallel Distributed Cognition

Connectionism is a paradigm of cognitive science that aims to explain cognitive abilities in terms of a complex network of simple computational units. It states a theory of information processing that relies on parallel computations on sub-symbols via a network of simple computational units. Higher functions, i.e. representation and transformation of information, emerge from the statistical properties of distributed activation patterns (Medler, 1998).

The Artificial Intelligence side of the connectionist movement gained crucial momentum with the publication of *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (Volume 1 and Volume 2) by Rumelhart et al. (1986) which led to its most successful implementation as *Artificial Neural Networks*. Neural networks were later proven to be indeed *universal approximators* (Hornik et al., 1989; Pollack, 1988), i.e. able to learn any function to an arbitrary degree, given an adequate learning regime, enough degrees of freedom (i.e. hidden units), and a deterministic relation between input and target. This holds even for simple single-layer networks with some non-linear activation function and no output squashing. These findings were a significant rehabilitation of the linear perceptron model after it had been proven to be incapable of learning the XOR function (Minsky and Papert, 1969).

2.2 Connectionism vs Classicist Model of Cognition

The principle of connectionism challenges the classic notion of the brain as *symbolic processor* which applies deterministic rules in order to manipulate symbols, each representing a complex piece of information (Newell and Simon, 1976). This idea of biological cognition is closely related to the Von-Neumann architecture of computers (Medler, 1998). Contrary to the symbolic approach, the connectionist framework allows graceful degradation in the face of noisy input, exceptions, or even the loss of single computational units, making the system very robust against damage and real-word situations. Attempts to reconcile both paradigms most notably include stating that connectionist models, especially Artificial Neural Networks, actually do emulate a symbolic processor via higher-level abstractions (Garson, 2015).

The difference between "emulating" and "learning" in turn has developed into one of the major criticisms of the connectionist paradigm. Connectionist systems model approximative associations, not deterministic rules, which seems to be in contrast to what is commonly thought of as higher reasoning and relatively clear-cut linguistic rules, e.g. consistent agreement patterns between verbs, subjects, and objects. The criticism is not that connectionist models cannot learn these functions, but rather that they just emulate the rules that could be described better by symbolic models. Much in contrast to symbolic models however, they are not able to systematically reuse the symbolic components of such a rule, such as the agent-patient relation between the arguments of a transitive verb (Garson, 2015). The claim that networks are incapable of explaining this kind of high-level cognitive function in biological brains has triggered a debate about *systematicity* in both connectionist and classicist models (Fodor and Pylyshyn, 1988) which continues to this day (Jansen and Watter, 2012). Outlining the intricacies of the whole discussion would exceed the scope of this section. See Garson (2015) for a concise summary of the debate.

One might conclude that Neural Networks are able to explain much of the same phenomena that classicists have tried to tackle, but with a smaller set of assumptions and axioms. Neural networks do not need rules for exceptions. If one regards Occam's razor as a sensible principle for choosing a paradigm, connectionist models probably win the battle between the two cognitive paradigms, not the least because they enabled the development of more robust systems than their symbolic counterparts in a vast array of applications. In other words, applicability has proved connectionism superior.

2.3 Connectionist Models of Grammar Learning

Standard feedforward connectionist models are stateless systems that perform a static mapping from input to output. Biological brains however are assumed to be not only conditioned on the current input, but on its context, notably the past which is embedded in an internal state (Williams and Zipser, 1989; Doya, 1995). Recurrent connections were recognized early as a crucial extension in order to account for this feature of cognition. One of the first approaches were Hopfield Networks (Hopfield, 1982) which aimed to gradually settle on stable states via recurrent associative dynamics. For a concise summary of the history of connectionist approaches to time-varying input, refer to Williams and Zipser (1989).

Backpropagation Through Time (BPTT) was soon discovered as a feasible approach to the computational complexity of learning deep architectures, providing the learning engine for the recently described Recurrent Neural Networks (RNN) (Rumelhart et al., 1988). The early success of RNNs was supported by the interesting activation patterns that emerged from them, for instance in natural language contexts.[1] Elman (1990) found that a simple RNN is able to unveil grammatical regularities in natural language sequences. Hidden activations exhibit a particular hierarchical pattern which are clearly visible after the application of hierarchical clustering for each input. This technique starts from every neural state being a cluster and recursively aggregates them into larger clusters via a distance measurement between their centers until there is only one big cluster left. The result is a tree of neural states, called a *dendrogram*. Tabor (1994) found that maximizing the distances between separate clusters as a measure of distances leads to the most distinct clusters.

Elman (1990) found that the hierarchical structures of the dendrograms reflect common concepts of morphosyntax and semantics, as described by linguists. For example, nouns are separated from verbs, the verb cluster is split into transitive and intransitive clusters, and nouns are structured into sensible semantic classes.

[1] Due to the high computational effort and constrained resources, a low-level pattern analysis was probably the only way to give them any practical justification.

This work was followed by Elman (1991) who trained a recurrent language model on a corpus of synthetic English sentences with a vocabulary of 23 words and a restricted grammatical variance, the purpose of which was to pose difficult situations for learning number agreement. Tabor (1994) claims that these early results prove that distributional patterns provide a solid base for semantics and that linguistic classification hierarchies can be encoded via proximities between points in a continuous metric space. Similar insights from such empirical investigations of grammar learning via RNNs are reported by Christiansen and Chater (1999) and Morris et al. (1998).[2] The empirical evidence of latent meaning embedded in patterns of co-occurrences had probably had an important influence on the recent advancements of distributed word embeddings which build on that very idea (Collobert et al., 2011).

It should be noted that these observations were drawn from English samples only. It seems likely that the syntactic peculiarities of other languages, specifically synthetic and agglutinating ones, would challenge the universal tone of their conclusions. Furthermore, the criticism of missing systematicity in connectionism (see Section 2.2) has not yet been solved by RNNs and the inclusion of a persistent state. Despite seemingly intelligent learning behavior, RNNs have yet to be proven to learn truly abstract rules that can be generalized to unseen constellations of input symbols. Advancing research in order to approach this goal has been one of the major motivations for the work in this thesis.

2.4 Issues of Modern Applications of Connectionism for NLP

Whereas synthetic tasks and scalpel-like analysis had been the norm until the late 1990s, it seems that there has been a gradual shift from analytical to empirical research which focuses on utilizing vast amounts of complex data to train generic architectures on abstract tasks and evaluates mainly on high-level performances. Halevy et al. (2009) argue that the abundance of data is what has driven the biggest advancements in Machine Learning for NLP, especially for hard tasks such as speech recognition and machine translation. For these problems, training data has become abundant and thus much cheaper than for simpler but less routine problems, such as part-of-speech tagging and parsing. Christoph Manning noted (Manning, 2015):

> *Recently at ACL conferences, there has been an over-focus on numbers, on beating the state of the art. Call it playing the Kaggle game. More of the field's effort should go into problems, approaches, and architectures. [...] I would encourage everyone to think about problems, architectures, cognitive science,*

[2] See also Garson (2015) for a summary of examples.

> *and the details of human language, how it is learned, processed, and how it changes, rather than just chasing state-of-the-art numbers on a benchmark task.*

The negative sentiment about the abandonment of low-level understanding efforts is shared by Joseph Reisinger who bemoans blind applications of generic systems on natural language problems[3]:

> *I get pitched regularly by startups doing "generic machine learning" which is, in all honesty, a pretty ridiculous idea. Machine learning is not undifferentiated heavy lifting, it's not commoditizable like EC2, and closer to design than coding. The Netflix prize is a good example: the last 10% reduction in RMSE wasn't due to more powerful generic algorithms, but rather due to some very clever thinking about the structure of the problem;*

Similarly, Michael Jordan states in a Reddit AMA (Ask Me Anything)[4] that he does not believe that NLP problems are best solved by generic systems:

> *Although current deep learning research tends to claim to encompass NLP, I'm (1) much less convinced about the strength of the results, compared to the results in, say, vision; (2) much less convinced in the case of NLP than, say, vision, the way to go is to couple huge amounts of data with black-box learning architectures.*

Manning (2015) summarizes his idea about the proper motivation for AI research:

> *Artificial intelligence requires being able to understand bigger things from knowing about smaller parts.*

In the opinion of the author, this is not only what AI research should be about, but what should drive research in general.

Weston et al. (2015) had stated a similar notion about a year before Manning when they introduced a synthetic data set of toy tasks to kickstart future question-answering research that would hopefully be more systematic and less driven by industrial purposes. Scaling down the experimental complexity and converting to tightly controlled problem setups is what they call an escape from "the local minima in algorithm space" that have been haunting the machine learning community.

[3] http://thedatamines.com/post/13177389506/why-generic-machine-learning-fails
[4] https://www.reddit.com/r/MachineLearning/comments/2fxi6v/ama_michael_i_jordan

2.5 Scaling Down: Toy Tasks of Natural Language Understanding

Richardson et al. (2013) propose a scalable method of retrieving open-domain crowd-sourced data sets of stories and questions in order to advance studies on high-level machine comprehension of natural language. The questions are in multiple-choice format and designed to be understandable by a 7-year old child. The answers depend solely on the given fictional stories, not on external knowledge.

On a similar notion, synthetic training corpora were used to concept-prove the first memory networks (Weston et al., 2014; Sukhbaatar et al., 2015). These data have since been applied to prove the sanity of new memory-based approaches (Grefenstette et al., 2015) while purposely leaving issues of data scarcity and algorithmic fine-tuning to the future.

The data used in this work is inspired by Weston et al. (2015) who propose a framework of synthetic toy tasks in order to train Question-Answering (QA) systems. A data set of document-question-answer triples is generated via a generative game engine, comprising 20 different tasks of increasingly difficult demands for logical reasoning, e.g. fact retrieval, entity relations, negation, deduction, and inference. Each task describes small environments which are connected through spatial relations and traversed by agents. These agents have a fixed set of actions at their disposal, e.g. moving around themselves or other objects.

While the application of memory networks has been proven effective, the framework of synthetic tasks has attracted some criticism. For example, Hermann et al. (2015) note several practical problems of this approach with respect to the scalability to real-life applications. Synthetic data sets are naturally restricted to a limited domain, due to the limitations of its vocabulary and generative grammar. The only feasible way to expand the topical scope is to use huge amounts of natural training data which are very expensive to annotate. This annotation can be performed by unsupervised training, creating knowledge graphs from the document on-the-fly via syntactic and semantic analyzers. Building on this idea, the group proposes an alternative way of creating annotated training data in a semi-automatic fashion. Real-language documents are transformed into templates, which in turn are multiplied against a vocabulary to produce almost arbitrary amounts of annotated document-query-answer triples. The proposed way of semi-automatic preprocessing however depends on sophisticated third-party tools, such as for coreference resolution, entity detection, and parsing of syntax and semantics. Adding more experimental variables to the setup obfuscates the responsibilities of different modules for the final performance and thus exacerbates low-level interpretations.

It is important to note that resorting to synthetic toy tasks is a historically dangerous endeavour. Early AI research was widely criticized for being unable to meet the combina-

torial explosion connected to real world problems, precisely because they were only able to handle low-complexity cases (Lighthill, 1973). Being able to handle simple data does not mean that the same phenomena are handled the same way in real-life experiments. A well-defined and reproducible methodology of modeling and measuring is indispensable for this approach to bear fruit, or else approaches based on toy tasks might suffer the same fate as first-generation AI: being unpractical and, worse, cognitively implausible.

Chapter 3

Recurrent Neural Networks

This chapter will describe the framework of Recurrent Neural Networks (RNNs) and present the algorithm for the extensions we apply in this thesis, namely Long Short-Term Memory (LSTM) and bidirectionality.

Section 3.1 reviews the recurrent framework of neural networks and discusses its benefits for NLP tasks.

Section 3.2 introduces one of its most popular implementations, Long Short-Term Memory Networks, by reviewing its motivations and applications, followed by a detailed description of the algorithm.

Section 3.3 describes the details of training RNNs for sequence classification, including a parameter-wise optimization algorithm that has been proven beneficial for recurrent models.

Section 3.4 describes a powerful extension to the recurrent framework: bidirectionality.

3.1 The Recurrent Framework

This section briefly outlines the motivation and history of Recurrent Neural Networks (RNNs) and provides a brief glimpse over their different branches (Section 3.1.1).

We give an overview of their technical limitations to explain why, despite their conceptual advantages for Natural Language Processing (NLP), RNNs had long succumbed to the model of Feedforward Multilayer Perceptron (MLP) (Section 3.1.2).

Furthermore, we discuss the deficits of MLPs for NLP and explain how recurrent models are a better fit for the problems of NLP (Section 3.1.3).

3.1.1 Introduction

Recurrent Neural Networks (RNN) were proposed in late 1980s and early 1990s as a solution to a nagging problem of feedforward Feedforward Multilayer Perceptron (MLP) models: that lack of sensitivity to the order of inputs (i.e. time), and thus a lack of memory (Werbos, 1988; Elman, 1990).

The basic idea of RNNs is that the hidden layer features a weighted connection to itself. This recurrent connection is traversed both during forward and backward propagation and allows the hidden units to carry information to later steps, thus forming a *memory layer*.

Definitions:

- f : smooth, bounded, nonlinear function, e.g. logistic sigmoid or hyperbolic tangent (tanh)
- $x \in \mathbb{R}^M$: upstream layer as vector of size M
- $W_{xh} \in \mathbb{R}^{N \times M}$: weight matrix of size $N \times M$ for connection from upstream layer to hidden layer
- $b_h \in \mathbb{R}^N$: bias of size N for affine transformation of hidden layer

The hidden layer $h \in \mathbb{R}^N$ is then computed as follows:

$$h = f(W_{xh}x + b_h) \tag{3.1}$$

The state of an RNN layer, in its simplest form, just adds a second linear mapping from the previous RNN state before applying the activation function, thus adding the following definitions:

- $t \in \mathbb{N}$: current time step
- $W_{hh} \in \mathbb{R}^{N \times N}$: weight matrix of size $N \times N$ for recurrent connection from hidden layer of previous time step to hidden layer of current time step

The RNN hidden layer $h_t \in \mathbb{R}^N$ is then computed as follows:

$$h^t = f(W_{xh}x^t + W_{hh}h^{t-1} + b_h) \tag{3.2}$$

Retaining information across several sequential inputs is a crucial ability of biological cognition. It allows modeling temporal correlations and distant dependencies without strict boundaries which is a corner stone of human natural language processing.

Though networks with a memory-like capability had been discussed before, notably Hopfield Networks (Hopfield, 1982), a scalable implementation had stayed elusive until a new learning regime was discovered in the late 1980s, the *backpropagation algorithm* (Rumelhart et al., 1986). This method could easily be generalized to a setup with shared parameters across several unfolded layers of the network. This reformulation became known as Backpropagation Through Time (BPTT) and has become one of the most popular ways of training RNNs, alongside other gradient-based methods (Williams and Zipser, 1995; Hochreiter and Schmidhuber, 1997).

RNNs have since evolved from history compression, such as in Elman Networks, Jordan Networks, and Multiplicative RNNs (Sutskever et al., 2011), to sophisticated memory controllers, such as Long Short-Term Memory Networks (Hochreiter and Schmidhuber, 1997), Gated Recurrent Units (Cho et al., 2014), Gated Feedback RNNs (Chung et al., 2015), as well as attention-based models such as Neural Turing Machines (Graves et al., 2014) and Memory Networks (Weston et al., 2014).

3.1.2 Limitations and Solutions

Despite the efficiency of the backpropagation method, RNNs have long suffered from severe practical limitations, mainly caused by the training method itself. Bengio et al. (1994) provided one of the first extensive studies on why long-term dependencies are difficult to model. The main culprit became known as the *vanishing gradient* problem which states that Backpropagation Through Time (BPTT) and gradient descent optimization are not able to sufficiently account for the responsibilities of distant inputs.

The problem of unstable gradients is well studied (Glorot and Bengio, 2010; Hochreiter et al., 2001; Pascanu et al., 2013). One of the most successful solutions, the LSTM algorithm, was proposed by Hochreiter and Schmidhuber (1997) who tackled the issue by the help of special gating units that control access to the memory layer via second-order connections (see Section 3.2).

A downside of LSTM units is that they replace the memory layer with a complex subgraph of layers and connections and an increase of the number of trainable parameters. More parameters not only slow down training but also require more data and longer training runs in order to converge. This was probably the reason why, despite being an effective solution to a crippling problem, research on LSTM networks had trouble entering the field of large-scale industry until a few years ago, and had thus long been denied the benefits of its financial support.

3.1.2.1 The Problem of Narrow Scopes

Feed-forward MLP models work strictly off input representations with fixed dimensions. If the number of features is known, their representations can be concatenated to a large so-called *projection layer* where each region corresponds to one feature.

In the field of NLP, one can thus simply cast the time dimension of natural language input to a fixed spatial representation, effectively restricting the scope of the network to a fixed window of words. For example, a sentence can be represented by a concatenation of word representations, and a word can be represented by a concatenation of its morphosyntactic features (Collobert and Weston, 2008). The assumption of a narrow scope resembles the Naive Markov Property in which future states are thought to depend only on the immediately preceding state, not earlier ones. Despite this limitation, MLP models led to important initial successes in NLP (Bengio et al., 2003; Schwenk and Gauvain, 2005).

However, many NLP tasks, such as document classification or language modeling, highly depend on flexibility with respect to the lengths of their input sequences, for which a fixed-window approach is either inconvenient or unacceptable (Goldberg, 2015).

A solution to this problem that sticks to the MLP network model is *Continuous Bag-Of-Words* representation proposed by Mikolov et al. (2013b). This method maps a variable number of input representations to their sum or a (weighted) average, thereby discarding information about the order. An alternative model proposed by the same group, the *skip-gram* model, reverses the problem to the prediction of a window of words, based on a given focus word. Both models were shown to be highly efficient and powerful in terms of the representations they produce (Mikolov et al., 2013a), despite the fact that they trade potentially useful positional information for larger context scopes.

The crippling problem of missing word order becomes evident in tasks like sentiment analysis, where it is crucial to know the relative position of the operator “not” with respect to the word it modifies (Goldberg, 2015).

3.1.3 Recurrent Language Processing

The recurrent framework provides a number of conceptual and practical benefits for Natural Language Processing (NLP), compared to non-recurrent approaches. Speech recognition is one of the most prominent applications of Neural Networks where respecting the given form of the data, an unbounded stream of raw sensory information, payed off by yielding vastly better results (Graves et al., 2004). But other more symbolic language processing tasks benefited as well.

For example, motivated by the success of connectionist MLP-based language models (Ben-

gio et al., 2003; Schwenk and Gauvain, 2005), Mikolov et al. (2010) introduce RNN language models as a further alternative to n-gram-based backoff language models. RNNs are able to use larger contexts without depending on fixed-size layers for the input window and the projection. When trained on the same subset of the Wall Street Journal corpus and applied to a speech recognition system, the RNN models produce 18% fewer word errors with a perplexity reduction of up to 50%. Further experiments by Sundermeyer et al. (2013) showed that the extension of RNNs by LSTM networks can leverage 8% perplexity improvement compared to standard RNNs.

Williams et al. (2015) recently confirmed the advantage of RNN language models over 5-gram models by training them on different training set sizes of Wikipedia, spoken and written news in English. Both models show a steady decline in perplexity in random subsets of 0.25 to 8 billion words with a vocabulary of 770,000 words. However, compared to the 5-gram model, the RNN starts and ends at much lower perplexities (ca. 82 to 68, compared to ca. 159 to 120) while maintaining a constant model size. Larger RNN models were able to further push down the perplexity to 42.4. Even at the far end of 4096 hidden neurons, the number of RNN parameters (541 million) stayed below a third of the comparable 5-gram model with 1,740 million parameters and a final perplexity of 66.9. Of course, as with all neural network models, the expressive power of RNNs comes at a high computational cost. The proposed model was trained for over 14 days on a high-performance GPU, compared to just 30 minutes for the 5-gram model.

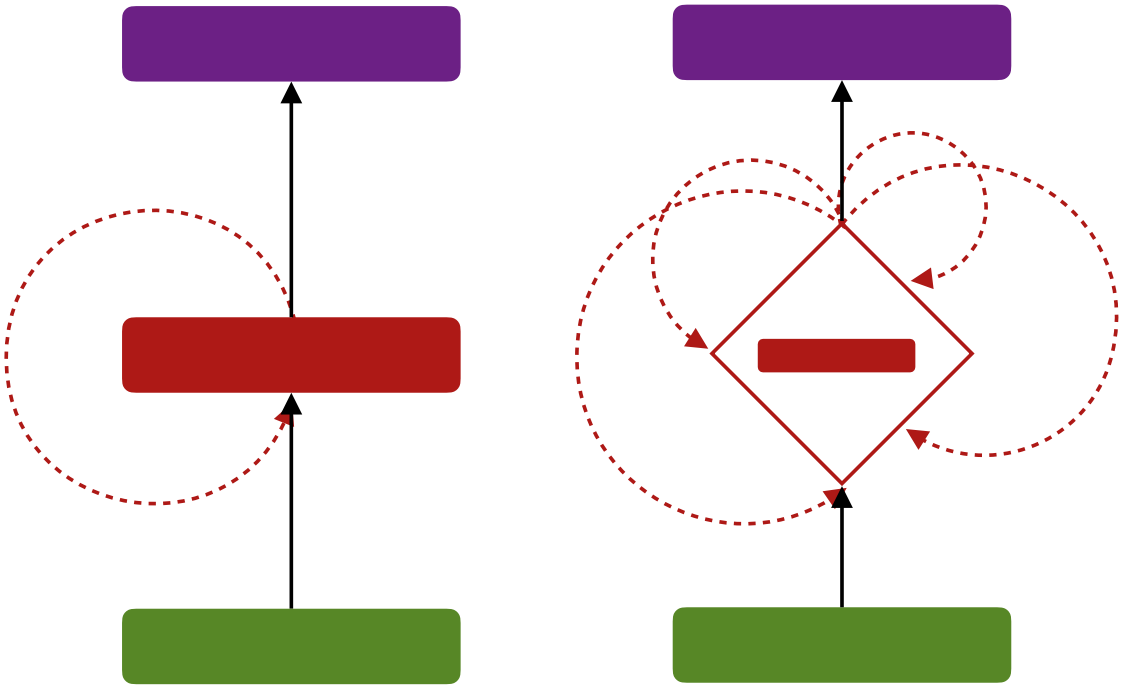
Though simple RNN models have been proved successful for NLP, the most important breakthroughs with this technology have been achieved after augmenting the recurrent connections with second-order control mechanisms, for example via Long Short-Term Memory (LSTM) units, as described in Section 3.2.

3.2 Long Short-Term Memory

Long Short-Term Memory (LSTM) networks were developed as a solution to the vanishing gradient problem (Hochreiter and Schmidhuber, 1997). The basic idea is to redefine the recurrent connection between memory layers to an identity mapping without weights and thus without gradients. This mechanism was adequately termed a *memory carousel*.

Instead, the weighted recurrent connections are outsourced to a new set of controller units that control fuzzy operations for reading, writing, and deleting cell contents at different stages of the sequence. Since the gates themselves are controlled by weighted connections, an LSTM implements a form of differentiable access to single memory cells. Figure 3.1 shows a schematic comparison of an LSTM unit and an RNN unit.

The role of gates as controller units makes LSTM a special case of a *second-order recurrent*



(a) Schematic RNN topology

(b) Schematic LSTM topology

Figure 3.1: Schematic comparison of a simple Recurrent Neural Network (RNN) unit and Long Short-Term Memory (LSTM) unit (without the optional peepholes). The upstream layer (blue) feeds into the hidden state (red), and optionally further downstream (purple). Recurrent connections are marked with a dotted arrow. In a simple RNN, the hidden layer is directly connected to itself. In an LSTM unit, the hidden layer does not directly control itself. Instead, it is governed by external units with their own recurrent connections.

neural network. Second-order connections enhance the weighted incoming activation by a second weighting factor in order to compute the new unit (Giles and Maxwell, 1987; Goudreau et al., 1994; Monner and Reggia, 2012). Giles et al. (1992) show that second-order connections allow RNNs to learn finite state automata.

Figure 3.2 visualizes the topology of the connections of an LSTM block across two time steps.

Despite this second-order property, LSTM units expose the same interface as first-order layers, and can thus serve as a drop-in replacement for any component in a neural network.

LSTM networks have been applied to modeling acoustic signals, both for input (Sak et al., 2014; Weng et al., 2014; Graves et al., 2013b; Maas et al., 2014; Graves and Jaitly, 2014) and output (Wu and King, 2016), as well as handwriting recognition (Frinken et al., 2012b) and optical character recognition (Breuel et al., 2013). LSTM models have also succeeded in solving a variety of NLP tasks, such as word-level language modeling (Sundermeyer et al., 2012), character-level language modeling (Karpathy et al., 2015), handwriting recognition (Frinken et al., 2012b, Graves and Schmidhuber (2008)), named entity recognition (Hammerton, 2003), speech recognition (Graves et al., 2004, Graves et al. (2013b), Sak et al. (2014)), and even lip-reading (Wand et al., 2016).

3.2.1 Architecture

A vanilla LSTM block exposes a standard layer interface to upstream layers (via the block input) and downstream layers (via the block output). The block itself is composed of a subnet of units with the following recurrent connections:

- From block output to the three gates and to the block input
- From the cell layer to the forget gate and to the input gate (recurrent peepholes)

The gates assume the role of second-order multiplicative weights and control to which degree information enters the block (**input gate**), if and how it is retained across time states (**forget gate**), and to which degree it exits the block (**output gate**).

In its basic configuration, an LSTM unit has four layers to which trainable connections lead (three gates and one block input), compared to a simple RNN unit with just one hidden layer. Each of these four layers computes its own sum of incoming weighted connections. Thus, instead of $M * N + N * N$ parameters for a simple RNN layer, a single LSTM block introduces a total of $4 * (M * N + N * N)$ trainable parameters. If peepholes are used, this number is increased by an additional $3 * N$.

According to Hochreiter and Schmidhuber (1997), despite this increase, the maximum computational complexity of LSTM networks per update step is $O(n^2)$, the same as

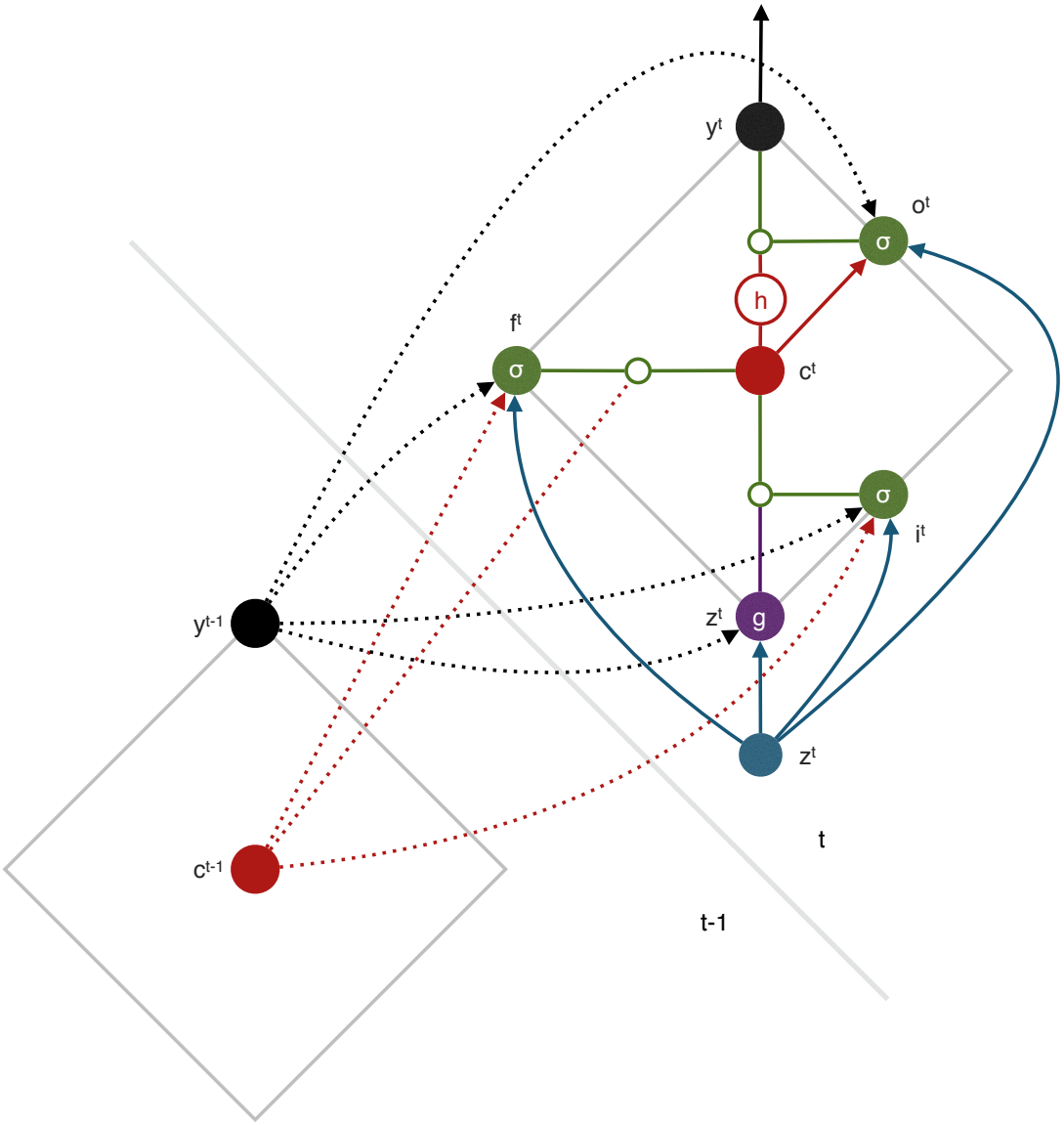


Figure 3.2: A schematic visualization of the connections within an LSTM unit over two time steps. The labels correspond to the layer names defined in Section 3.2.3. Color-filled nodes represent layers: blue is upstream layer, purple is block input, red is memory cells, green is gates, black is block output. White nodes with a colored border indicate an intermediate activation step (red border) or a pointwise multiplication by an adjacent gate (green border). Arrows represent weighted connections of which dotted arrows are recurrent. Arrows that converge to the same node indicate a summation of the weighted input at that node. σ denotes sigmoid activation, g and h denote *tanh* activation.

the complexity of simple recurrent networks that use Backpropagation Through Time (BPTT).

Name	Definition
input	input to current level from upstream level
block input	summation layer at LSTM block input
input gates	input gate layer
forget gates	forget gate layer
memory cells	memory cell layer
output gates	output gate layer
block output	multiplication layer at LSTM block output
output	network output after last level

Table 3.1: Definitions of layer names.

For brevity, clarity and consistency, we will refer to the layers within an LSTM unit by the names described in Table 3.1, and whenever possible in that order.

Note that the memory cell layer is only allowed to communicate to units of the same block, i.e. to itself (via the identity mapping), to its gates (peepholes), and the block output. Communication outside the block is restricted to the block output.

3.2.1.1 Forget Gates

The first LSTM algorithm by Hochreiter and Schmidhuber (1997) was not able to reset the memory cells unless the sequence featured a special token that could be learned to trigger the deletion, or unless they were split into subsequences and reset manually. This deficit led to the inclusion of a special gate unit, the *forget gate*, which was shown to accelerate and improve convergence (Gers et al., 2000). It has since become a mandatory extension to the LSTM algorithm.

The forget gate is plugged right into the connection of the memory carousel and controls the memory transfer from one time step to the next. If the forget gate is set to 1, it retains the cell content. If it is set to 0, it deletes the cell content.[1]

3.2.1.2 Peepholes

Gers et al. (2002) extended the LSTM topology with *peepholes*, i.e. point-wise weighted connections from the memory cells to the gates.

[1] A better name might have been *remembering gate* or *recall gate*.

The motivation is to give each memory cell a "voice" in deciding whether they should be retained, overwritten, or carried further downstream in the next time step. Without peepholes, if the output gate is shut down, there would be no recurrent influence from cells to the next steps' gates. Peepholes thus allow more precise counting and keeping track of time intervals.

Two of these peephole connections are recurrent (to the forget gates and to the input gates). They are most commonly implemented as weight vectors, not matrices, in order to let each cell only control its own gate, not the gate of other memory cells.

Though peepholes seem to be useful in tasks that require precise counting of distances, they do often not seem to justify the increased number of parameters they introduce. Studies show that they are useless to detrimental for some NLP tasks, such as recognition of speech and handwriting (Greff et al., 2015) as well as language modeling (Karpathy et al., 2015). One of the most thorough studies of the usefulness of peepholes was performed by Breuel (2015a) who found that peepholes do not help and sometimes even negatively affect the performance on MNIST image classification and Optical Character Recognition. It is therefore common to exclude peepholes from setups, especially since they require a whole new set of trainable parameters. Consequently, the RNN library for Torch offers a `fastLSTM` module where the removal of peepholes allows a more efficient training (Léonard et al., 2015).

3.2.2 Alternatives

LSTM units expand the network with a heap of new parameters to train and have attracted some criticism for the seeming ad-hoc nature of its additional components that do not provide a perfect solution to the problem of the vanishing gradient (Józefowicz et al., 2015). More specifically, LSTM gates are designed to provide merely a bypass for the error across temporal steps. As long as a gate retains a value close to 1, it delays the gradual dissolution of the gradient, but does not prevent it completely (Chung et al., 2014).

In order to address the issue of high computational cost, Cho et al. (2014) introduce a gated network architecture based on the idea of LSTM gating mechanism, called Gated Recurrent Units (GRU). GRUs are essentially a simplification of LSTM that combines the forget gate and input gate into one "update gate" and replaces the output gate by a "reset gate" that controls only the recurrent connections to the block input. It further merges the memory cell layer and the block output layer in favor of one single exposed memory layer. GRUs thus sacrifice both the controlled encapsulation of the memory content and the independency between the inclusions of current and past input in favor of a radical reduction of computational complexity.

Despite these simplifications, GRU networks have been shown to perform similar to LSTM networks in terms of sequence modeling (Chung et al., 2014). Józefowicz et al. (2015) found that GRU units seem to outperform LSTM units on almost all tasks, but also note that the differences vanish if the LSTM forget gate bias is initialized to 1. The group also confirms the less important role of the output gates of LSTM units, supporting the idea of GRUs to remove them completely. The idea of coupling forget gates and input gates has been explored by Greff et al. (2015) who show that this computational simplification has no significant impact on the overall performance. However, Józefowicz et al. (2015) note that the forget gates and input gates are the most important gates for certain tasks, so the question of whether they should be combined in order to save parameters is not yet settled.

As an extension to the LSTM architecture, Depth Gates RNNs (Yao et al., 2015) introduce an additional gate unit that links the memory cells of two different levels of the network in order to control the downstream information flow. The group reports a perplexity reduction from 117 to 96 on language modeling of the Penn Treebank data set.

Another noteworthy alternative to the LSTM architecture that tackles the vanishing gradient problem is Clockwork RNNs (Koutník et al., 2014). The basic idea is to split up the hidden component of a standard RNN into separate modules, each of which receives input from different time steps. This modification is shown to outperform even LSTM networks on speech data classification.

3.2.3 Algorithm

The equations in this section are based on the vectorized LSTM equations in Greff et al. (2015). They include forget gates and peepholes.

Each layer type assumes one state for each time step t. These layer states are declared as follows, roughly following the order of their computations:

- x^t : vector of *inputs* from the upstream layer
- z^t : vector of *tanh*-activated **block inputs**
- i^t : vector of *sigmoid*-activated **input gates**
- f^t : vector of *sigmoid*-activated **forget gates**
- c^t : vector of non-activated **memory cells**
- o^t : vector of *sigmoid*-activated **output gates**
- y^t : vector of *tanh*-activated **block outputs**

For each vector of activated values, there is a corresponding vector of non-activated values marked by a top bar, e.g. $\bar{z}$. These raw values are needed for backpropagation which depends on the derivative of their respective activation function.

The sizes of the dimensions of the parameter tensors are then defined as follows:

- $M = |x|$
- $N = |z| = |i| = |f| = |c| = |o| = |y|$

The parameter tensors are then defined as follows:

- $W_i, W_f, W_o, W_z \in \mathbb{R}^{N \times M}$: **input weights** (inputs → gates and block inputs)
- $R_i, R_f, R_o, R_z \in \mathbb{R}^{N \times N}$: **recurrent weights** (block outputs → gates & block inputs)
- $p_i, p_f, p_o \in \mathbb{R}^N$: **peephole weights** (memory cells → gates)
- $b_i, b_f, b_o, b_i \in \mathbb{R}^N$: **biases**

3.2.3.1 Forward propagation

The following computations are iteratively executed from the beginning of the sequence $t = 1$ to the end of the sequence T. The vectors of y^0 and c^0 are initialized with zeros.

The $\odot$ operator denotes the pointwise product of two vectors. The activation functions g and h are implemented as *tanh*.

Block input:

$$\bar{z^t} = W_z x^t + R_z y^{t-1} + b_z \tag{3.3}$$
$$z^t = g(\bar{z^t}) \tag{3.4}$$

Input gates:

$$\bar{i^t} = W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i \tag{3.5}$$
$$i^t = \sigma(\bar{i^t}) \tag{3.6}$$

Forget gates:

$$\bar{f^t} = W_f x^t + R_f y^{t-1} + p_f \odot c^{t-1} + b_f \tag{3.7}$$
$$f^t = \sigma(\bar{f^t}) \tag{3.8}$$

Memory cells:

$$c^t = f^t \odot c^{t-1} + i^t \odot z^t \tag{3.9}$$

Output gates:

$$\bar{o^t} = W_o x^t + R_o y^{t-1} + p_o \odot c^t + b_o \tag{3.10}$$
$$o^t = \sigma(\bar{o^t}) \tag{3.11}$$

Block output:

$$y^t = o^t \odot h(c^t) \tag{3.12}$$

3.2.3.2 Backpropagation Through Time

Backpropagation Through Time (BPTT) (e.g. by Werbos, 1988) is a generalization of the backpropagation algorithm (Rumelhart et al., 1988) that unfolds an RNN to a DNN with shared weights between the recurrent layers. Within this unfolded network, normal backpropagation can be performed to compute the parameter tensor gradients for each time step. For each parameter tensor, its gradients from all time steps are then summed in order to compute the final gradients (see Section 3.2.3.3).

As in normal backpropagation, the partial derivatives of the sequence loss function E with respect to each upstream layer (*deltas*) depend on deltas from downstream layers, as defined by the chain rule.

Equation 3.13 presents the definitions of two example deltas for the LSTM block output layer y^t and the memory cell layer c^t.

$$\delta_y^t = \frac{\partial E}{\partial y^t} \qquad \delta_c^t = \frac{\partial E}{\partial c^t} \tag{3.13}$$

The following equations define the implementation of the chain rule to compute the deltas of all layers of an LSTM unit at all time steps. The computations are iteratively executed from the end of the sequence $t = T$ to the beginning of the sequence $t = 1$. All δ_*^{T+1} are initialized with zeros.

The $\odot$ operator denotes the pointwise product of two vectors. Here, it is used only for the multiplicative units that perform the second-order weighting by the gates. The notation δ_l^t denotes the delta of layer l at time step t.

The sum for the block output deltas (Equation 3.14) includes the summands for the posterior gate deltas. These are present in Greff et al. (2015), but not in Graves (2012) for an undisclosed reason. Since they are technically necessary for a computationally sound backpropagation that respects all forward connections, we include them as well.

Block output deltas (Δ^t denotes the weighted delta from the downstream layer):

$$\delta_y^t = \Delta^t + R_z^T \delta_z^{t+1} + R_i^T \delta_i^{t+1} + R_f^T \delta_f^{t+1} + R_o^T \delta_o^{t+1} \tag{3.14}$$

Output gate deltas:

$$\delta_o^t = \delta_y^t \odot h(c^t) \odot \sigma'(\bar{o}^t) \tag{3.15}$$

Memory cell deltas:

$$\delta_c^t = o^t \odot \delta_y^t \odot h'(c^t) + f^{t+1} \odot \delta_c^{t+1} + p_i \odot \delta_i^{t+1} + p_f \odot \delta_f^{t+1} + p_o \odot \delta_o^t \tag{3.16}$$

Forget gate deltas:

$$\delta_f^t = \delta_c^t \odot c^{t-1} \odot \sigma'(\bar{f}^t) \tag{3.17}$$

Input gate deltas:

$$\delta_i^t = \delta_c^t \odot z^t \odot \sigma'(\bar{i}^t) \tag{3.18}$$

Block input deltas:

$$\delta_z^t = \delta_c^t \odot i^t \odot g'(\bar{z}^t) \tag{3.19}$$

Input deltas (in case of another hidden layer upstream):

$$\delta_x^t = W_z^T \delta_z^t + W_i^T \delta_i^t + W_f^T \delta_f^t + W_o^T \delta_o^t \tag{3.20}$$

3.2.3.3 Gradients

The following equations define the computation of the final weight gradients from the layer deltas. The computations can be performed in any order.

The $\otimes$ operator denotes the outer product of two vectors. The $*$ notation in the subscripts can be any of $\{z, i, f, o\}$.

Inputs $\rightarrow$ gates:

$$\delta_{W=\sum_{t=0}^{T} \delta^t} \otimes x^t \tag{3.21}$$

Block output $\rightarrow$ gates (recurrent):

$$\delta_{R=\sum_{t=0}^{T-1} \delta^{t+1}} \otimes y^t \tag{3.22}$$

Memory cells $\rightarrow$ gates (peepholes):

$$\delta_{p_i} = \sum_{t=0}^{T-1} c^t \odot \delta_i^{t+1} \tag{3.23}$$

$$\delta_{p_f} = \sum_{t=0}^{T-1} c^t \odot \delta_f^{t+1} \tag{3.24}$$

$$\delta_{p_o} = \sum_{t=0}^{T} c^t \odot \delta_o^t \tag{3.25}$$

Biases:

$$\delta_{b=\sum_{t=0}^{T} \delta^t} \tag{3.26}$$

3.3 Training Recurrent Neural Networks

3.3.1 Error and Backpropagation

As in language modeling (LM), the loss function is defined as the cross entropy between the target and the log-softmax normalization of the output layer. In other words, the cross entropy is the entropy of the target distribution o with respect to what the model expects.

Definitions:

- $P_o(k)$: probability function on element k of the output vector
- $P_y(k)$: probability function on element k of the target vector
- K : size of the output vector (and target vector)
- $n \in \mathbb{N}^+$: index of current sample

The current sample's cross entropy $H(P_y^n; P_o^n)$ of the target distribution P_y^n of sample n with respect to the model's actual output distribution P_o^n is then defined as follows:

$$H(P_y^n; P_o^n) = -\sum_{k=1}^{K} P_y(k) log_2 P_o(k) \tag{3.27}$$

In case of any form of batch processing, the evaluation measure is the average cross entropy over N samples:

$$H(P_y^{\sum}; P_o^{\sum}) = \frac{1}{N} \sum_{i=1}^{N} H(P_y^i; P_o^i) \tag{3.28}$$

Note that "batch processing" just means that the errors of several outputs are accumulated in order to boost training speed and encourage better generalization.

In a language model, this averaging is usually performed over all tokens of one sequence sample, and then additionally over the averaged errors of several samples (which is the common notion of "batch processing"). In a many-to-one sequence classifier, only the latter sample averaging is performed because there is only one error per sample. For each batch (or single sample), this loss computed from the cross entropy is then Backpropagated Through Time to the beginning of the sequence (see BPTT algorithm in Section 3.2.3.2). As this backpropagation applies for all samples of the batch, all samples are required to have the same length.

The following section outlines the algorithm of stochastic gradient descent and explains the principles of per-parameter adaption of the global learning rate.

3.3.2 Stochastic Gradient Descent and Learning oRate Decay

Stochastic Gradient Descent (SGD) is a first-order optimization method that computes the parameter gradients on one or a few training samples. The rate of descent depends on a global learning rate which is applied equally to the gradients of all parameters.

Definitions:

- i : index of current iteration (i.e. optimization step)
- Θ : vector of parameters as flat view on the concatenation of all parameter tensors
- δ_{Θ^i} : gradient of parameters Θ at iteration i (as defined in Section 3.2.3.3)
- α : global learning rate (optionally subject to a global decay rate)

In stochastic gradient descent, the tensor of parameters Θ is then updated as follows:

$$\Theta^{i+1} = \Theta^i - \alpha * \delta_{\Theta^i} \tag{3.29}$$

Since the learning rate in vanilla SGD is globally fixed, it is unable to adapt to changes in the training run. For example, it can be beneficial to reduce the impact of an update when the model is only supposed to change slowly, e.g. at later stages of the training run and/or when the losses fall below a certain threshold. A quick hack is thus to use a global *learning rate decay* that modifies the learning rate according to a schedule.

However, there are two more issues with this simple implementation of SGD:

Sparse data and infrequent features

In large networks and with sparse data, different parameters likely correspond to different features of the data set. Here, rare features should be treated with different importance than frequent features. Applying the same global learning rate to all these gradients does not account for their different contributions to the prediction error (Duchi et al., 2011).

Saddle points in the error surface

Training large Deep Neural Networks (DNN) (including Recurrent Neural Networks (RNN)) is a problem of optimizing complex non-convex error functions. Their error surfaces commonly suffer less from local minima than from saddle points in the error curvature, i.e. areas that are surrounded by slopes that are facing to different directions (up and down) and are hard for SGD to escape (Dauphin et al., 2015).

A solution to this problem is resorting to second-order optimization functions that utilize pseudo-curvature information to escape saddle points, for example Hessian-free optimization (Martens and Sutskever, 2011). However, these optimizers are very expensive in time and space and thus problematic for large parameter spaces (Kingma and Ba, 2014).

3.3.3 Parameter-Wise Learning Rate Adaptation

In order to address the lack of sensitivity of SGD for feature frequencies in sparse data, Duchi et al. (2011) introduce the idea of a parameter-wise update of the learning rate in form of a family of Adaptive Subgradient Methods (*AdaGrad*). These optimization methods adapt the learning rate for each parameter such that rare features trigger larger updates, and frequent features trigger smaller updates. This is achieved by accumulating squared gradients by which the learning rate is divided for each gradient. AdaGrad was used by Dean et al. (2012) to stabilize training on deep networks with billions of parameters. Pennington et al. (2014) show that this technique can be applied to natural language data and helps producing high quality word embeddings.

However, AdaGrad suffers from monotonically decreasing learning rates due to the constant accumulation of squared gradients and can thus cause the optimization speed to die down in long training runs. As a solution to this problem, Zeiler (2012) modified AdaGrad to *AdaDelta* which maintains a running average of the squares from windows of past gradients (root mean squares). The algorithm does not require a global learning rate.

A similar idea, the *RMSprop* algorithm, allows keeping the global learning rate and its decay. RMSprop was developed independently by Tieleman and Hinton (2012), derived from the *rprop* algorithm (Riedmiller and Braun, 1993) in order to work with stochastic gradient descent on minibatches. This optimization method is claimed to approximate the advantages of second-order optimizers and help escape saddle points in the error curvature. As an additional advantage, RMSprop helps mitigate the problem of careful hyperparameter tuning, especially for momentum calibration, and thus provides beneficial learning dynamics from the start (Dauphin et al., 2015).

The idea of RMSprop is to update a cache of element-wise factors r^i which are applied to the gradient vector δ_{Θ^i} before each optimization step at $i+1$. The gradients are then scaled by r^i and multiplied by the global learning rate α:

$$\Theta^{i+1} = \Theta^i - \alpha * \frac{\delta_{\Theta^i}}{\sqrt{r^i + b}} \tag{3.30}$$

b is a small smoothing constant to avoid division by zero, commonly 1e−8. Note that this notation is mathematically equivalent to scaling α for each gradient before multiplication but more efficient to implement.

From Equation 3.30 it is obvious that the higher the respective value in the cache, the smaller the effective learning rate for that parameter. On the other hand, parameters that are updated less often with smaller values receive a higher effective learning rate in order to boost their adaptation speed.

RMSprop implements the scaling factor by maintaining a cache of moving averages of the root mean squared (RMS) gradients:

$$r^i = \gamma * r^{i-1} + (1 - \gamma) * (\delta_{\Theta^i})^2 \tag{3.31}$$

The computation of the scaling factor is performed element-wise over the vector of parameters. Note that squaring the gradients effectively disposes of any directional information.

A constant decay factor γ controls the influence of the previous cache value and is commonly in the range of 0.9, 0.99, or 0.999.

Kingma and Ba (2014) extend RMSprop by Adaptive Moment Estimation (*Adam*). Adam disconnects the parameter update from the gradient scaling and instead computes it directly via a mechanism with further hyperparameters. The algorithm also allows a correction of the bias which could otherwise lead to unwanted large update steps. Since this algorithm introduces further tuning possibilities and does not seem to have been used and studied as extensively as RMSprop, we choose to use RMSprop.

3.4 Bidirectional Recurrency

Bidirectional Recurrent Neural Networks (BRNN) allow a recurrent network to access both past and future contexts for a current prediction, whereas standard RNNs just allow the former. Given a sequence of inputs, at each step in time, past and future contexts are fed into two parallel recurrent hidden layers, the *forward component* and the *backward component*. These two layers in turn are weighted and fed into the downstream layers of the network. This is mathematically equivalent to concatenation and subsequent weighting (Bahdanau et al., 2014) which is more modular and thus easier to implement (see Section 3.4.1).

In other words, the two bidirectional components are just parallel forks within the network. If the backward component were taken out of the equation, one would simply end up with

a standard RNN layer. This trick allows a complete, symmetric representation of both sub-sequences right and left from the current input, while still relating each input to its own target (Graves, 2012).

Bidirectionality in recurrent neural networks was first introduced by Schütze (1993) in order to facilitate a more context-sensitive part-of-speech labeling. The network implements a language model that predicts the middle word between its left and right 4-gram contexts, each of which feeds into its own sub-RNN in forward and backward direction respectively. The output layers of these sub-RNNs are then concatenated into the downstream layer. As a standard RNN, this architecture is trained via Backpropagation Through Time (BPTT). Schuster and Paliwal (1997) later proposed a similar idea that also relies on the concatenation of sub-RNNs and compared its effectiveness for phoneme classification to that of a naive merging of two networks. BRNNs have since been proven superior compared to unidirectional architectures in a vast array of applications, for example protein structure prediction (Baldi et al., 1999), generative time series reconstruction (Berglund et al., 2015), machine translation (Bahdanau et al., 2014), language modeling (Frinken et al., 2012a), handwriting recognition (Graves et al., 2009) and speech recognition (Graves et al., 2013a; Schuster, 1999; Maas et al., 2014).

3.4.1 Algorithm

The formula for computing the hidden state of a bidirectional RNN unit is a generalization of the one for a unidirectional RNN unit (see Equation 3.2). The main idea is to concatenate the hidden states of two unidirectional units, one forward component and one backward component, in order to yield one single hidden state which is fed to downstream network layers. The schematic visualization in Figure 3.3 shows how a bidirectional RNN unit works in the context of its upstream and downstream neighbors.

The following equations describe how the hidden layer of the two components are computed, by the example of a simple RNN (as formulated by Graves et al. (2013a)):

$$\overrightarrow{h}_t = f(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}) \tag{3.32}$$

$$\overleftarrow{h}_t = f(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}) \tag{3.33}$$

The output h_t of the whole bidirectional unit is then presented to the next layer as a concatenation of its two components:

$$h_t = \{\overrightarrow{h}_t; \overleftarrow{h}_t\} \tag{3.34}$$

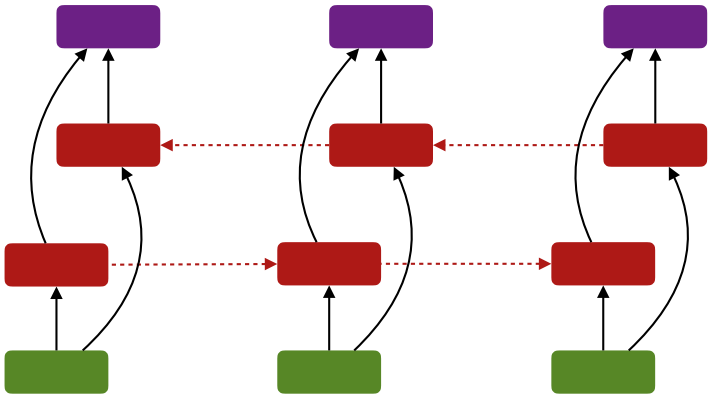


Figure 3.3: Schematic visualization of a bidirectional RNN. The upstream layer (blue) feeds into the hidden state (red), and optionally further downstream (purple).

During forward propagation, the activations of the forward component $\overrightarrow{h}_t$ and $\overleftarrow{h}_t$ of all time steps t are computed as follows (Graves, 2012):

for t = 1 to T **do**
    compute and store $\overrightarrow{h}_t$
for t = T to 1 **do**
    compute and store $\overleftarrow{h}_t$

In backpropagation, the forward component and the backward component depend on deltas from the respective other end of the sequence. Therefore, contrary to a standard RNN layer, we cannot simply compute downstream layer deltas on the fly while backpropagating through time. All T downstream deltas have to be computed first. This is no problem as long as the full sequence is known a-priori, as is necessary for BRNNs.

Given all T downstream deltas, deltas for the forward component $\overrightarrow{\delta}_t$ and for the backward component $\overleftarrow{\delta}_t$ are computed as such:

for t = T to 1 **do**
    compute $\overrightarrow{\delta}_t$ from downstream deltas
for t = 1 to T **do**
    compute $\overleftarrow{\delta}_t$ from downstream deltas

Applying this algorithm to a bidirectional LSTM can be achieved by simply redefining the forward component $\overrightarrow{h}_t$ and the backward component $\overleftarrow{h}_t$ as two parallel LSTM networks.

Chapter 4

Coreference Resolution

In Section 3, we presented Recurrent Neural Networks (RNN) as a tool for sequence classification. In the remainder of this thesis we apply this tool to the problem of Coreference Resolution (CR), a subtask of discourse analysis that requires profound semantic understanding of the context. This chapter provides an introduction to CR and outlines the core principles and frameworks.

Section 4.1 introduces the linguistic principles and basic terminology, and discusses the motivation of choosing CR as problem of natural language understanding.

Section 4.2 discusses the role of careful feature design and how it feeds into rule-based approaches of CR.

Section 4.4 discusses the application of rich features in machine learning systems that are supposed to solve many of the shortcomings of traditional deterministic approaches.

4.1 Introduction

Coreference Resolution (CR) is the process of resolving the chain of mentions that refer to the same real-world entity in the discourse. A mention is typically a noun phrase (proper, nominal, or pronominal) that refers to a person, an object, or an abstract concept (Kobdani, 2012).

CR is closely related to the problem of Entity Linking, with the difference that CR groups mentions into equivalence classes (coreference entities) such that all members of the class are linked to each other, not to an external referent. The equivalence relation between coreferent entities thus satisfies the properties of *reflexivity*, *symmetry*, and *transitivity*, as defined by Van Deemter and Kibble (2000):

Def. CR: α_1 and α_2 corefer if and only if Referent(α_1) = Referent(α_2).

A special case of CR is the problem of Anaphora Resolution (AR) where the problem is reduced to finding the **antecedent** of an **anaphor**, most often a pronominal phrase. According to Van Deemter and Kibble (2000), the crucial difference between CR and AR is that anaphoric relations are *irreflexive*, *non-symmetric*, and *non-transitive*:

Def. AR: An NP α_1 is said to take an NP α_2 as its anaphoric antecedent if and only if α_1 depends on α_2 for its interpretation.

One could assume from these definitions that CR is merely an extension of AR. If AR were applied to non-anaphoric entities and coreferential antecedents were identified iteratively, one would be let to think that it naturally satisfies the definition of full CR.

However, these definitions state a clear distinction between CR and AR: Not all coreferential relations are anaphoric, and not all anaphoric relations are coreferential. An example of the latter are bound anaphora:

Whenever a solution emerged, we embraced it.

Since there is technically not a specific *solution* that is being referred to, the anaphoric relation here is not coreferential by definition. Therefore, from a strict cognitive linguistic perspective, CR and AR merely exhibit some overlap, but cannot be described as generalization or special case of the other.

As Zhekova (2013) points out, the practical goal of AR is not to interlink all existing antecedents, but to identify the entity that the anaphor refers to. According to Mitkov (1999), the task of AR is completed when one such relation is found.

In contrast, full CR is completed only when all references to a referent have been identified. For the purposes of a computational model, breaking down CR to its subtasks, e.g. the identification of a valid pair of mentions, is a valid and necessary approach.

The following sections and chapters will elaborate on these practical considerations.

4.1.1 Mention-Pair Model

Clark and González-Brenes (2008) categorized Coreference Resolution (CR) models into two broad categories:

- cluster-based models which evaluate the entire CR chain at once
- pairwise models which resolve only two mentions at a time

While the cluster-based approach seems conceptually more sound, it also poses a difficult inference problem for the system. Therefore, the *mention-pair* model of CR has so far prevailed as the more successful approach.

The basic idea of the *mention-pair* model of CR is to reduce the task of resolving the chain to a binary classification of whether two proposed mentions are indeed coreferent or not (Soon et al., 2001). The training data is thus presented as a set of positive and negative samples for mention pairs.

In case of Anaphora Resolution (AR), all antecedents can be detected by iteratively applying the binary classifier to a pair of candidates and one anaphor. After each comparison, the winner is retained, eventually yielding a sorted list of best candidates (Mitkov, 1999).

4.1.1.1 Deficits and alternatives

However, the mention-pair model lacks two major properties that are necessary to fully account for the definition of CR as states in definition 4.1:

1. It does not account for transitivity (i.e. it does not resolve arbitrary chains of mentions).
2. It does not account for interdependencies across several mentions.

Rahman and Ng (2014) describe alternatives to the mention-pair model that are supposed to solve these issues: the *mention-ranking model*, the *entity-mention model*, and their own model of *cluster-ranking*. However, these more sophisticated solutions come at higher computational complexity, and none of them solve all the issues at once. Thus, the mention-pair model has largely prevailed as the base model of more complex systems. Subsequent problems, such as clustering mentions to a chain, are deferred to later modules down the CR pipeline (Zhekova, 2013). CR systems based on this approach have retained their position among top performing systems ever since (Ng, 2010), even for multiple languages (Pradhan et al., 2012), and are still being investigated (Uryupina and Moschitti, 2015).

4.1.2 Understanding Language by Resolving Coreferences

Coreference Resolution (CR) is a good example of a well-defined and narrow task that nevertheless relies on profound Natural Language Understanding (NLU).

Kehler and Rohde (2013) performed a set of psycholinguistic experiments in order to uncover the underlying mechanisms of pronoun interpretation. They found that probabilistic-driven experiences based on world-knowledge are a crucial component of resolving and expecting coreference relations. These findings motivate a data-focused attempt to solving the task. If the conclusions drawn from these experiments are true, it is likely that a generic machine learning approach to modeling co-occurrences between corefering entities is able to perform basic CR.

However, as most top-performing systems require rich morphosyntactic and semantic features (see Section 4.2), it is clear that resolving a coreference chain strongly benefits from information besides distributional patterns. CR can therefore be seen as a proper subtask of Question-Answering (QA) which requires deep NLU. This claim is supported by the fact that CR was shown to be implicitly resolved as part of a larger QA problem (Hermann et al., 2015).

As a narrowly defined subtask of NLU, CR brings a range of benefits:

1. **Quality**: CR is a well-established NLP task and comes with a reasonable amount of annotated training data of high quality (e.g. from CoNLL-2012).
2. **Control**: The data can be reduced to entail a controlled subset of linguistic phenomena.
3. **Measurable**: Certain subtasks of CR, such as AR, allow straightforward evaluation and analysis.
4. **Transferable**: It can be formulated as a problem of Question-Answering, as will be shown in Section 5 and Section 6.

In subsequent chapters, we will provide evidence for this claim and show how CR can indeed leverage a deeper understanding of a Recurrent Neural Network (RNN).

4.2 Features of Coreferences

The most successful contemporary systems for Coreference Resolution (CR) rely on rich lexical, morphosyntactic, and/or semantic features (Zhekova, 2013). Consequently, a great deal of research effort has been invested into finding low-cost features that provide the most benefit (e.g. Ng and Cardie, 2002a). Common examples for such features are (Kobdani, 2012):

- Distance between the mentions
- Sizes of mentions
- Substring matches between mentions
- Case
- Number
- Gender
- Person
- Part-of-speech
- Semantic classes

These features can be annotated either manually or via third-party tools, with the obvious tradeoffs between quality and scalability.

4.3 Rule-Based Systems of Coreference Resolution

The problem of Coreference Resolution (CR) has traditionally been a problem not only of feature design, but also of defining the proper set of rules over these features, inspired by the linguistic relationships between mentions. Summarizing about two decades of development in the field of rule-based CR systems would exceed the scope of this thesis. We instead refer to the overviews given by Zhekova (2013) and Kobdani (2012).

Several studies suggest a somewhat discouraging future for a generic Machine Learning (ML) approach. It seems that even simple rule-based systems are able to outperform all unsupervised approaches and most supervised approaches, as long as rich features are available (Bengtson and Roth, 2008; Haghighi and Klein, 2009; Haghighi and Klein, 2010; Raghunathan, 2010).

Despite the successes of rule-based systems, these investigations suggest this approach suffers from a well-known problem of such systems that will ultimately prove to be an uphill battle: poor generalization to unseen events, such as unseen grammatical constellations or new domains. From this problem follows poor scalability, especially for new languages, since each new domain requires its own set of rules and annotated data. Furthermore, Raghunathan et al. (2010) note that most rule-based systems rely on confidence scores computed from a single function over these local features. This causes inaccuracies when many low-precision features are allowed to win against few high-precision features, for example mere proximity and morphological agreement against semantic relation. They therefore suggest an extension to the rule-based approach utilizing feature ranking which is shown to successfully tackle a new range of specific linguistic phenomena. This new method provided the winning system in the shared task of CoNLL-2011 (Lee et al., 2011). Recently, this approach has been successfully reconciled with a ML mechanism, yielding top performance even across different domains and languages (Lee et al., 2013).

A less benevolent interpretation of this rule-ranking-based solution is that it is a mere ad-hoc modification of a fundamentally flawed paradigm. The underlying issue of rule-based systems is that they depend heavily on a proper feature design and that they are not able to derive new latent features that the engineer did not think of.

Furthermore, they lack a mechanism to account for complex synergies between these features, latent or not. As these synergies are difficult to define manually, they have to be learned in order to dynamically adjust the importance of latent and non-latent properties of the data.

In other words, a scalable and robust solution to the Coreference Resolution Problem is a natural problem for Machine Learning.

4.4 Overview of Machine Learning Approaches to Coreference Resolution

In the previous section, we discussed how rich features and sophisticated rules are the cornerstone of top-performing CR systems, yet how they also suffer from severe scalability issues. For a number of NLP tasks, Machine Learning (ML) is used to tackle the problem of data sparsity and ensure proper generalization on unexpected events. Therefore, it seems that ML is a natural choice to tackle the issue of CR, which is fundamentally about a profound semantic understanding of the context.

Supervised and unsupervised systems differ in the way they distribute the complexity mass between the features and the algorithm. Unsupervised systems are at the extreme end of models that require as little human effort as possible. The overall motivation is to move away from explicit feature engineering, both driven by expensive manual and unreliable third-party-driven methods.

This section will provide a short overview of several approaches along this direction. For an extensive overview of CR systems across several machine learning methods, refer to Olsson (2004).

4.4.1 Supervised Models of Coreference Resolution

Among the early applications of ML for CR is the system proposed by Soon et al. (2001) who rely on richly annotated data to resolve general noun phrases. Bengtson and Roth (2008) prove that a simple classifier on top of strong features can easily catch up with more sophisticated models.

Quite contrarily, (Durrett and Klein, 2013) try to fight the "uphill battle" by avoiding extensive hand-crafted heuristics and show that easy gains can be achieved via simple surface-level features, defined by simple templates for mention lengths, mention heads, first/last words within the mention and words immediately preceding and following a mention. These shallow properties are powerful enough to implicitly capture linguistic features which would otherwise have to be defined manually. They show that while mention-pair compatibility on the level of morphosyntax and discourse can be modeled well, compatibility solely derivable from semantics between mentions remains difficult. For example, they show that such a simple system relies heavily on exact head match of the two mentions and performs notably worse in case of anaphoric (i.e. no head match) relations. They conclude that "many mentions in this category" can only be correctly resolved by exploiting world knowledge", such as hypernymy, synonymy, number, and gender.

For a further review of neural-network-based approaches to supervised CR, refer to Section 6.1.

4.4.2 Unsupervised Approaches

Since annotated corpora require a lot of human effort to create, recent years have seen a rising interest in training unsupervised CR systems via a range of clustering approaches with little or no prior annotations. The basic idea is to bootstrap high-level inferences by providing low-level distributional information (Poon and Domingos, 2008).

Interpreting the problem of CR as a clustering problem allows tackling some of the shortcomings of the mention-pair approach (Rahman and Ng, 2014).

With a similar motivation, Kobdani et al. (2011) completely refrain from pre-annotated data and utilize statistical word associations for unsupervised auto-labeling a CR training corpus. This automatically labeled data is then used to train a supervised decision tree model to identify coreference chains. The resulting system significantly outperforms other unsupervised approaches. As a caveat, the authors report a high sensitivity for the quality of the corpus, especially the number of singletons which skew the bias, though this can easily be evaded by simply increasing the volume of raw training data.

Ng (2008) interpret the problem of resolving full coreference chains as a problem of unsupervised expectation-maximization clustering. Their system depends on a small handcrafted set of features, both linguistic (gender, number, semantic class), and lexical (e.g. string match, appositive). Their system compares favorably to other unsupervised systems (e.g. Haghighi and Klein, 2007), but does not provide state-of-the-art performance. Furthermore, they note that extending this system merely by adding more features is difficult, since the learning regime requires non-overlapping features.

For an overview of the main obstacles in designing prior clustering methods for CR, refer to Clark and González-Brenes (2008).

Chapter 5

Predictive Anaphora Resolution

This chapter applies a shallow recurrent Long Short-Term Memory (LSTM) network to the task of head noun prediction for Anaphora Resolution (AR). We build on the methodology of Elman (1990) whose simple Recurrent Neural Network (RNN) language model was able to extract grammar patterns from toy sentences. We will advance this model to a modern LSTM setup and analyze the emerging patterns from a classification task. The analyses of these experiments lay the groundwork for further low-level analysis of RNNs. We present a method of visual inspection that can give systematic insight into the behavior of single neurons within small networks.

The *Torch* code for reproducing the results in this chapter is available at:
https://github.com/kaumanns/DeepKaspar

Section 5.1 provides an overview of related work on supervised AR based on neural networks.

Section 5.2 explains how the task of AR is cast to the problem of sequence classification over a vocabulary.

Section 5.3 describes the process of creating a toy corpus from a generative definite clause grammar (DCG).

Section 5.4 explains the LSTM network configuration and experimental setup.

Section 5.5 discusses the results of the performance evaluation.

Section 5.6 concludes the experiments with a first visual analysis of single neuron states, retrieved from the layers in selected samples of the test set.

Section 5.7 discusses limitations and offers suggestions for further improvements on the experimental design.

5.1 Related Work

This section reviews related work of supervised Anaphora Resolution (AR) based on neural networks.

Recasting the problem of AR to a classification model has long been a common methodology. According to Olsson (2004), the earliest work on AR with a neural network classifier is Connolly et al. (1997) trained by the backpropagation algorithm proposed by Mitchell (1997).

Motivated by this early application, Stuckardt (2007) trained a multi-layer neural network to classify anaphoric relationships. The system is based on feature tuples generated by a rule-based processing of pairs of anaphora and antecedent candidates. It is shown that this approach performs as well as or better than decision tree models or naive Bayes models that had been common so far for this task.

On a similar notion, Orasan and Evans (2011) show that detecting the animacy of an anaphor and its antecedent noun phrase candidates is an important pre-filtering step for further resolution efforts. Their nearest-neighbor-based machine learning system reaches near-human performance, contrary to the rule-based alternative. Both utilize WordNet and a number of hand-encoded features, such as numbers of animate/inanimate WordNet senses and ratios of animate/inanimate pronouns, in order to identify animacy of isolated linguistic entities (i.e. without surrounding context).

Another notable example is the recent work by Küçük and Yöndem (2015) on a rule-based approach for pronoun resolution in Turkish, a language for which little knowledge base data is available. Their system relies on a small set of morphosyntactic and lexical heuristics, yielding a recall of 73.7% and a precision of 91% on children's story texts. Though these rules are still hand-engineered, they prove that a minimal set of features, which are possibly learnable by a neural network over larger amounts of data, can kickstart an AR system even on little data.

To the knowledge of the author, no system exists that applies a standard RNN architecture to the task of antecedent head prediction.

5.2 Casting Anaphora Resolution to Sequence Classification

This section explains how the task of Anaphora Resolution (AR) is cast to the problem of sequence classification over a vocabulary.

5.2.1 Meaning Function

This section explains how we define the meaning function by which we model the anaphoric relationship between two mentions.

We reduce the problem of AR to the prediction of the head noun of the antecedent. This allows us to frame the task as a classification problem over the corpus vocabulary.

Kumar et al. (2015) argue that most NLP problems, including word prediction, can be cast to a supervised question-answering problem where the training data is composed of *document-query-answer* triples. A question-answering model would thus maximize the following conditional probability:

$$P(Answer|Document, Query) \tag{5.1}$$

Analogously, we define the conditional probability of the head of the antecedent given the context as such:

$$P(Antecedent|Context) \tag{5.2}$$

The context spans the whole discourse, including the antecedent and the anaphor, as well as a special query token that separates the context from the query and cues the network to predict the antecedent.

Definitions:

- w_i : the head of the antecedent
- w_j : the anaphor (pronoun)
- n : the length of the document sequence
- $1 <= i < j <= n$
- q : the query token

The antecedent classifier maximizes the conditional probability of the antecedent head, given the context and the query:

$$P(w_i|w_1, ..., w_i, ..., w_j, q) \tag{5.3}$$

This task reminds one of a natural problem for a Memory Network (MemNN) (Weston et al., 2014) as it requires the retrieval of a specific element that occurred in the past.

However, we argue that a MemNN would clearly overshoot the problem. We know that RNNs, and especially LSTM networks, are perfectly able to solve the task of language modeling where the number of output classes corresponds to the vocabulary that makes up the sequence (Sundermeyer et al., 2012). We know further that RNNs are Turing-complete (Siegelmann and Sontag, 1995) and satisfy the conditions for the universal approximation theorem (Cybenko, 1989; Hornik, 1991).

Therefore, a recurrent LSTM network should be perfectly able to implement the classification function in Equation 5.3 and thus emulate a MemNN that retrieves a word from the past.

5.2.2 Data Design

As discussed in Section 5.2.1, our function implements a sequence classifier in order to solve a question-answering problem. We thus represent our data as a simple concatenation of context and answer where the query token serves as a prediction-triggering delimiter:

```
[context tokens] <query> <answer>
```

The context sequence can be of variable length. The query and the answer both have a length of 1 token.

A further design choice is whether we want to represent single tokens as symbolic 1-of-k representations or as distributed representations (word embeddings). Word embeddings can be trained from 1-of-k representations in two ways: either on the fly via a projection layer during training of the final task, or by pre-training a distribution model in an unsupervised fashion on a large unannotated data set. Embeddings trained by the latter method have been shown to boost the performance of natural language understanding systems, especially if these systems are trained on small data set (Collobert et al., 2011; Erhan et al., 2010). They therefore seem to be a mandatory extension for our setup. However, as discussed in Section 2.5, pre-training would introduce another layer of experimental complexity which could ultimately affect the quality of our analysis. In order to maintain a pure performance and allow unspoiled analysis, we stick to 1-of-k word representations.

5.3 Synthesizing Data

This section describes the process of creating a toy corpus from a generative definite clause grammar (DCG).

In order to maintain tight control over the complexity of our data, we implement a generative grammar that produces all possible combinations of its vocabulary. This is very similar to the approach described by Weston et al. (2015), where toy stories are generated from a restricted set of agents, environments, and actions.

5.3.1 Generative Grammar

A definite clause grammar (DCG) engine implemented in PROLOG exhaustively generates all variants that adhere to the following grammar.

Let

- upper case tokens be variables for members of their respective classes
- the subscripts i, j, and k index the referent of a mention
- the superscript G marks gender agreement of two units with the possible values $neut$, fem, and $masc$
- commas (,) separate alternative values for a lexical unit

Then the DCG grammar is defined as follows (see Section 9.1 in appendix for the full PROLOG code):

S → HUMAN$_i^G$ and HUMAN$_j$ <eos> their ANIMAL$_k$ VP PRON^G <q> HUMAN$_i$
S → HUMAN$_i$ and HUMAN$_j^G$ <eos> their ANIMAL$_k$ VP PRON^G <q> HUMAN$_j$
S → HUMAN$_i$ and HUMAN$_j$ <eos> their ANIMAL$_k^G$ VP PRON^G <q> ANIMAL$_k$
VP → ADVERB VERB
VERB → likes,loves
ADVERB → really, usually, very often, after a while
PRON → her^{fem}, him^{masc}, its^{neut}
ANIMAL → dog$_1^{neut}$, cat$_2^{neut}$
HUMAN → Mary$_1^{fem}$, Susie$_2^{fem}$, Frank$_3^{masc}$, John$_4^{masc}$, Peter$_5^{masc}$

As proposed by Hermann et al. (2015), we use a special query token (<q>) to indicate the position at which the question for the antecedent is asked. The token <eos> marks the end of the first sentence, but else has no special role in the upcoming experiments.

The resulting dataset contains 704 unique discourses, one per line. Examples:

```
Frank and John <eos> their dog very often likes him <q> John
Frank and Mary <eos> their dog really likes her <q> Mary
Frank and Mary <eos> their cat really loves its <q> cat
```

Crucial characteristics of these discourses are:

- Antecedents are always in the same positions with respect to each other.

- Antecedents have variable distances to the anaphor and the query token, separated by a variable-length adverbial phrase.
- Two adjacent HUMANs must have different referents (i.e. be different names), but they are allowed to have the same gender.

The resulting sequences are shuffled and split into training and test set, ensuring that the network is tested on unseen sequences. The split ratio is one of the experiment variables.

5.3.2 Features

The grammar described in the previous section was designed with a hierarchy of features in mind, each with a different degree of importance for resolving the anaphor.

Primary feature: animacy

HUMANs share the same two positions in the sequence, but HUMANs are always in different positions than ANIMALs.

Secondary feature: gender

While *animacy* is sufficient to resolve neutral pronouns, a further *gender* feature is necessary to separate *masculines* from *feminines*.

Tertiary feature: order

Even after *animacy* and *gender* are resolved to be one of the human genders, it may be necessary to resolve between two *masculines* or two *feminines* in terms of their *order*. Our system always assumes the last mention as the correct one.

In summary, the following features have to be learned by the network in order to be able to resolve all anaphora:

1. animacy $\in \{\text{animal}, \text{human}\}$
2. gender $\in \{\text{masculine}, \text{feminine}, \text{neuter}\}$ (*neuter* is equivalent to *animal* in this data set)
3. order $\in \{\text{first}, \text{last}\}$ (applies only when two *humans* of the same *gender* are present)

We will show that all these features play a role in the decision process, depending on the experimental setup.

5.4 Setup

This section explains the LSTM network configuration and experimental setup.

5.4.1 Architecture

We implement a recurrent many-to-one sequence classifier using an RNN toolkit for the *Torch* machine learning library (Léonard et al., 2015). Selected parts of the code, notably the preprocessing, are modifications from the character-level LSTM language model by Karpathy et al. (2015).[1]

Our setup is similar to a language model as it classifies the token sequence over its own vocabulary. The only difference is that there is only one prediction, deferred to the very last time step. Thus, there is no need of summing and averaging the prediction errors of the sequence, as is commonly done in language modeling.

Figure 5.1 visualizes the information flow in this recurrent network.

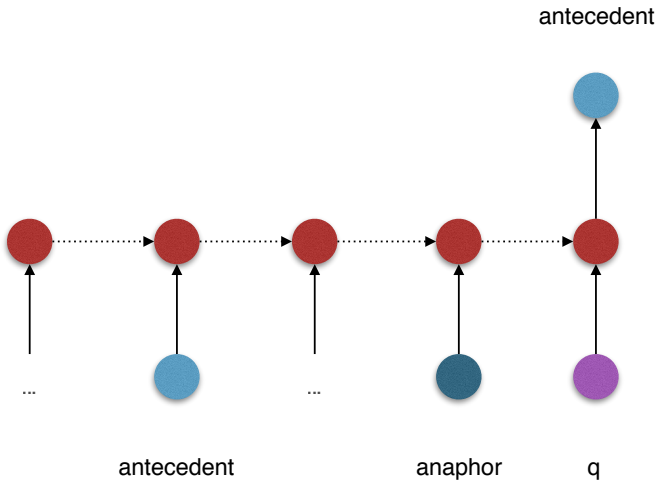


Figure 5.1: Visualization of the information flow in a recurrent many-to-one sequence classifier. Each arrow represents a full connection between the layers. Recurrent connections are marked by dotted arrows. The task is to predict the antecedent to the anaphor after the entire sequence has been read, including the query token $<q>$.

The recurrent toolkit features a variable-length input which eliminates the need for sequence padding. Each sample is presented as a *context-query-answer* triple where the context consists of the token sequence and the query token, resulting in the following form:

```
[context tokens, including antecedent] [anaphor] <q> [antecedent]
```

For examples, see Section 5.3.

[1] https://github.com/karpathy/char-rnn

The data is split into two sets: training set and validation set. Note that the validation set is not used for hyperparameter-tuning, therefore it technically corresponds to a test set.

The training set is shuffled after each epoch in order to desensitize the network for the order in which the samples are presented.

An LSTM unit contains two states which have to be cached in order to compute the recurrent connections: the memory cells and the block output. These states have to be reset after each sample in order to prevent unwanted state bleeding between the semantically independent samples.

5.4.2 Error and Backpropagation

We apply Backpropagation Through Time (BPTT) for backpropagating the sequence loss across the time steps to the beginning of each sample sequence (see Section 3.2.3.2). Truncation is not necessary because samples are presented one by one and have a fixed length. We only need to ensure that the network states are reset to 0 after each update.

As in language modeling (LM), the range of output classes spans the entire vocabulary of the corpus, not just the antecedents. The target token (the *antecedent*) is thus represented by the same 1-of-k encoding as the input tokens, requiring the sequence error to be defined as the cross entropy between the final prediction and the *antecedent* token at the end of each sample sequence (see Section 3.3).

5.4.3 Optimization

All experiments are run over a fixed number of 30 epochs. Each epoch comprises a full run through the training set and a full run through the test set. The epoch sizes thus depend on the respective set split ratio which is one of our experiment variables (see Section 5.4.5).

Each sample triggers an update of the network parameters (i.e. sample-wise online learning). Parameters are updated via RMSprop (Dauphin et al., 2015), a variant of stochastic gradient descent that adapts the learning rate per parameter (see Section 3.3.3). In addition to parameter-wise gradient adjustments, the global gradient is decayed by a factor of 0.95 each epoch, starting at epoch 10.

In order to keep the network activations pure and easily interpretable, we use neither speed enhancing techniques (such as minibatches, multithreading, or class layers), nor any form of active anti-overfitting precautions (such as regularization, momentum, or dropout).

5.4.4 Accuracy and Baseline

Definitions:

- $set \in \{train, test\}$: one of the split sets
- $1 \leq epoch \leq 100$
- acc_{set}^{epoch} : the accuracy of set at $epoch$
- $|set|$: number of samples in set
- o_i : the output layer at sample i
- y_i : the target class at sample i

The accuracy per epoch is defined as:

$$acc_{set}^{epoch} = \frac{|\{y_i | y_i = \text{argmax}_i o_i\}|}{|set|}$$

A trivial majority baseline would pick the token that appears most frequently in the contexts of the training set. However, it is safe to assume that a bug-free system would at least learn to choose randomly from the set of target tokens. We therefore define $V_{targets}$ as the subset of the vocabulary that contains only the target tokens. Then the random baseline accuracy for both training and test set is simply the average baseline accuracy over all targets:

$$acc_{base} = \frac{1}{|V_{targets}|} \tag{5.4}$$

Our target vocabulary spans 7 antecedents as target candidates. The random baseline is therefore ca. 14.29%.

5.4.5 Set Ratios

Let r be the size ratio of the training set ($train$) to the test set ($test$), in number of samples. Table 5.1 provides an overview of the set sizes for each ratio.

5.5 Evaluation

Performances are evaluated with respect to set ratio r and the size of the memory cell layer c. This evaluation setup is motivated by the following two assumptions:

Split ratio *train*/*test*	*train*	*test*
9/1	633	71
3/1	528	176
1/1	352	352
1/3	176	528

Table 5.1: Number of samples in training set and test set for different split ratios.

Split ratio train/valid	Baseline	1 hidden	2 hidden	3 hidden	4 hidden
9/1	.14	.57 / .49	.91 / .89	1.00 / 1.00	.99 / 1.00
3/1	.14	.43 / .41	.85 / .84	.99 / .99	.99 / .98
1/1	.14	.47 / .43	.64 / .65	.99 / .99	1.00 / .99
1/3	.14	.46 / .44	.85 / .83	.95 / .92	.94 / .85

Table 5.2: Accuracies for combinations of split ratio of training set and validation set (train/valid), as well as number of hidden neurons (hidden).

1. The smaller r, the harder the generalization problem.
2. The smaller c, the harder it is to learn the features necessary to resolve the anaphora.

Our goal is to measure to which degree the performance on the toy data set depends on these two variables. The resulting accuracies for different combinations of split ratio and hidden size are shown in Table 5.2.

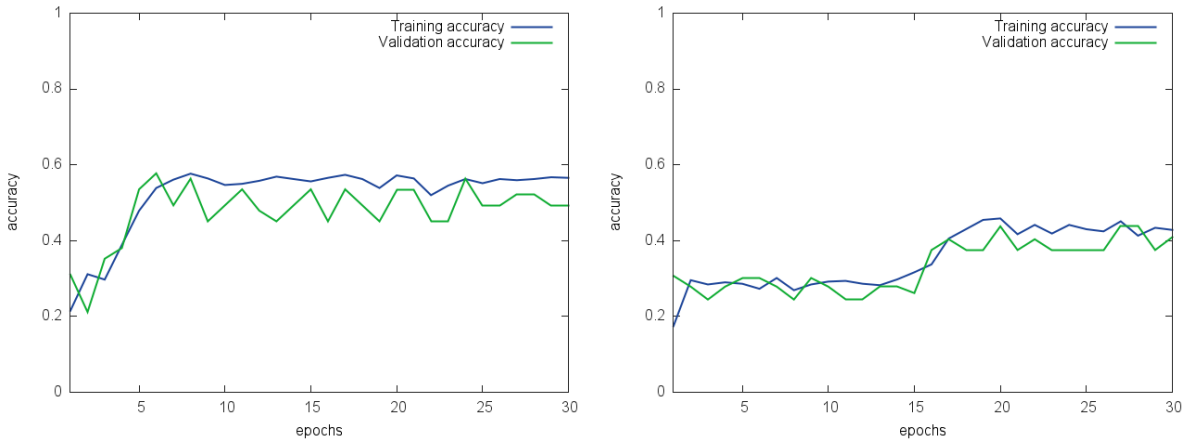
In general, more hidden nodes and higher split ratios mean better performance, but all configurations perform well above the random baseline, even with small hidden layers and extremely unfavorable split ratios.

The remainder of this section investigates how these performances develop in the course of the training. Recall that the network, in order to achieve perfect performance, has to be able to remember all three antecedents just in case any of them is referred to by the anaphor at the end of the sequence.

One single hidden neuron (Figure 5.2) performs remarkably well, even if the learning is a little rocky.

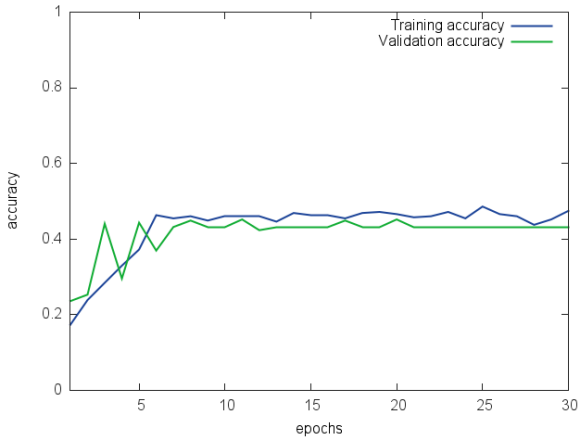
All split ratios exhibit a common pattern: the performance rises fast above baseline but soon hits a ceiling at ca. 50% for $r = 9/1$ (Figure 5.2a) and at ca. 40% to 45% for all other ratios (Figure 5.2b, Figure 5.2c, Figure 5.2d). All ceiling accuracies apply relatively equally to training and validation set, taking the oscillations for the former into account.

The low consistent accuracy indicates that the single memory cell learned a stable feature which minimizes the error, though it does not resolve all cases. The fact that training and test sets performed quite similarly indicates that this feature must be one that is inde-

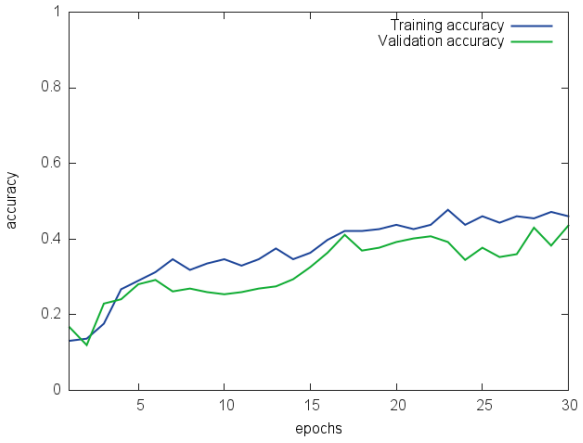


(a) $r = 9/1$

(b) $r = 3/1$



(c) $r = 1/1$



(d) $r = 1/3$

Figure 5.2: Accuracies for LSTM size 1 across different split ratios r of training set size to validation set size

pendent from the properties that distinguish the training samples from the test samples. This leaves only the anaphor itself.

It seems that this single-cell network did not learn a representation for each of the 7 antecedents, but only of the anaphor. This hypothesis can be tested by manually deriving the probability model that would produce the observed accuracies.

If the interpretation stated above is true, then the task of modeling $P(Antecedent|Context)$ (see Equation 5.2) is reduced to modeling the combined probability of the antecedent and the anaphor:

$$P(Antecedent \cap Anaphor) = P(Anaphor) * P(Antecedent|Anaphor) \tag{5.5}$$

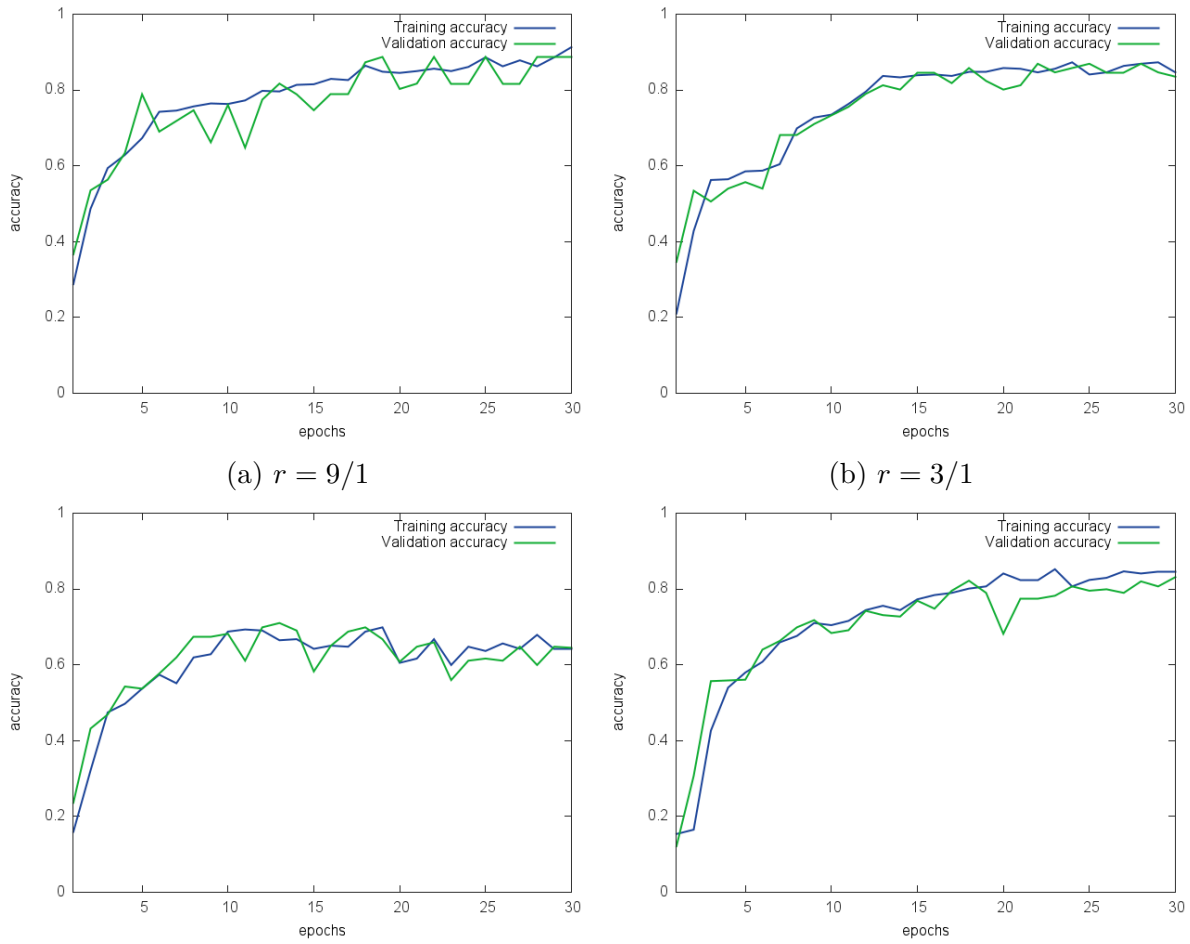
We further assume that the 7 antecedents are distributed equally in both training and validation set. If the network learned to represent the anaphor then it learned the probability $P(Anaphor)$ of each anaphor event:

$$P(its) = 2/7 \tag{5.6}$$
$$P(her) = 2/7 \tag{5.7}$$
$$P(his) = 3/7 \tag{5.8}$$
$$\tag{5.9}$$

As the network has no clue about the context, only the anaphor, it models $P(Antecedent|Anaphor)$ as a selection according to this distribution. Therefore, the model performance for picking the right antecedent can be predicted as the summation of three mutually exclusive events as follows:

$$P(Antecedent) = P(Antecedent \cap its) + P(Antecedent \cap her) + P(Antecedent \cap his) \tag{5.10}$$
$$= P(its) * P(Antecedent|its) \tag{5.11}$$
$$+ P(her) * P(Antecedent|her) \tag{5.12}$$
$$+ P(his) * P(Antecedent|his) \tag{5.13}$$
$$= (3/7) * (1/3) + (2/7) * (1/2) + (2/7) * (1/2) \tag{5.14}$$
$$= 0.4285714 \tag{5.15}$$

This is indeed roughly the performance we observe for the training runs in Figure 5.2. Small deviations from this predicted accuracy are probably due to the fact that the



(a) $r = 9/1$

(b) $r = 3/1$

(c) $r = 1/1$

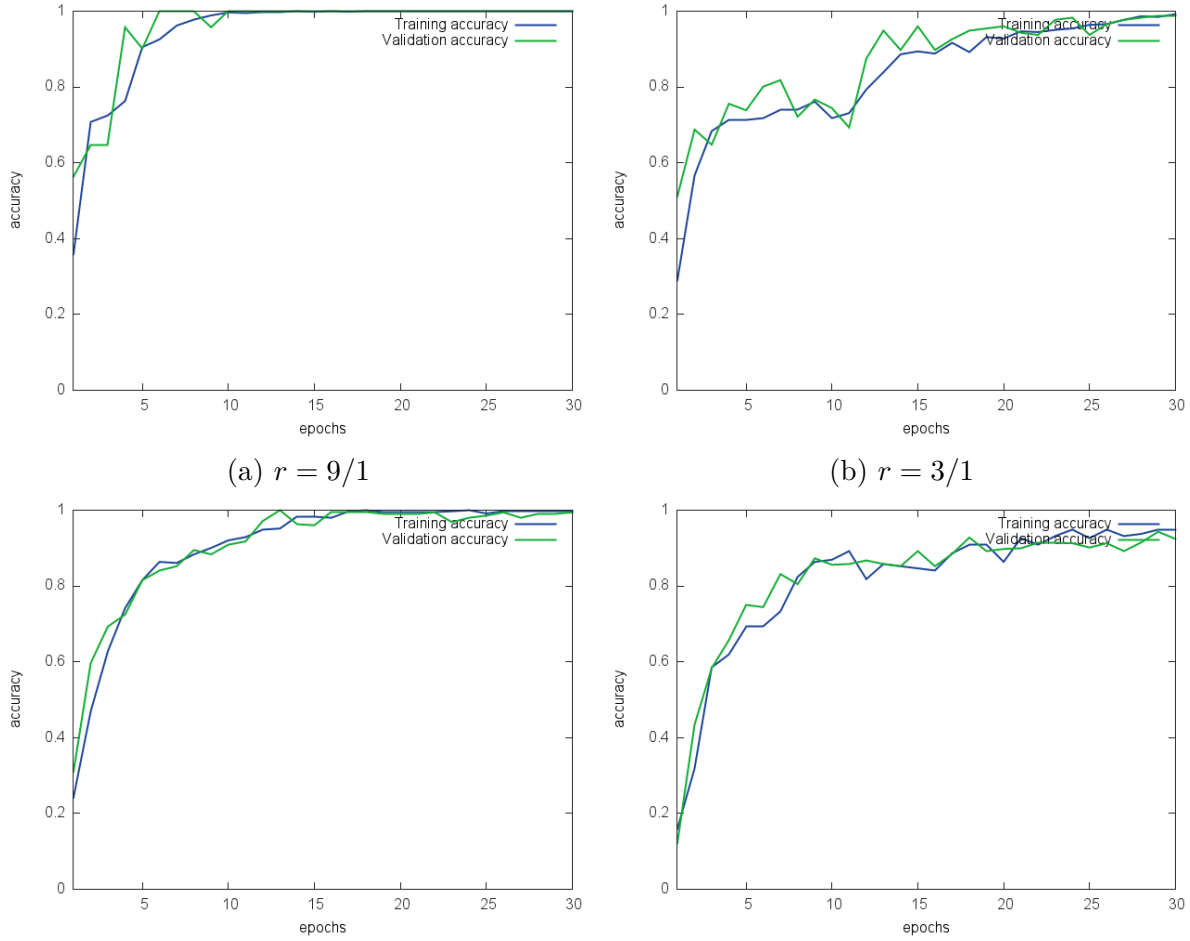
(d) $r = 1/3$

Figure 5.3: Accuracies for LSTM size 2 across different split ratios r of training set size to validation set size

antecedents are not perfectly equally distributed in each set. We thus conclude that the single-cell model learned to correctly represent the anaphor (i.e. *gender*), but not the antecedent from the context.

Two hidden neurons (Figure 5.3) continue the general narrative of rocky learning. However, this model converges to a much higher final performance, at least for higher split ratios (Figure 5.3a, Figure 5.3b). Interestingly enough, the performance at a 1/1 ratio experiences a short peak and then drops clearly never to recover (Figure 5.3c).

We see a much more stable and successful generalization for an even lower ratio, almost reaching the performance of higher splits (Figure 5.3d). This could be due to an unlucky distribution of samples in the 1/1 split, or conversely due to a lucky distribution in 1/3.



(a) $r = 9/1$ (b) $r = 3/1$

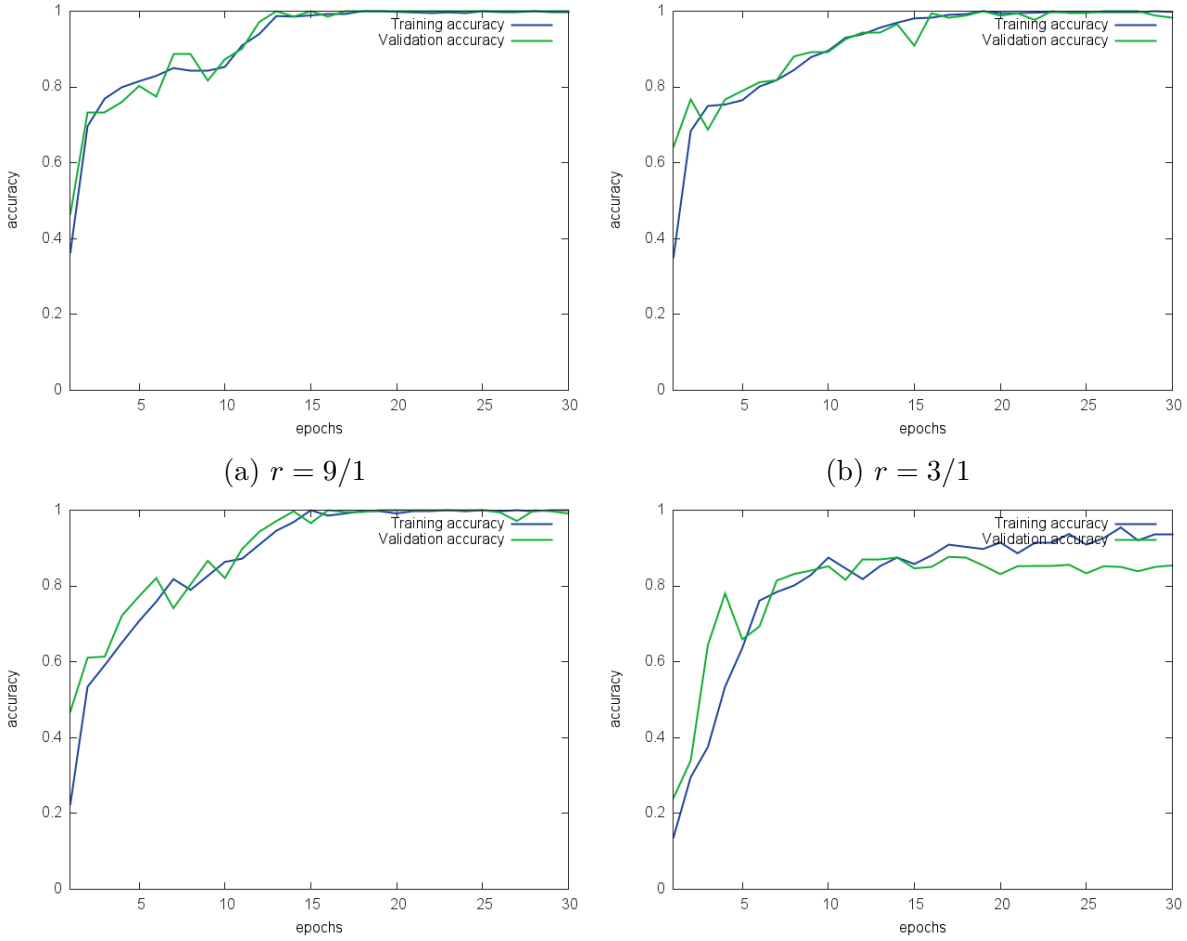
(c) $r = 1/1$ (d) $r = 1/3$

Figure 5.4: Accuracies for LSTM size 3 across different split ratios r of training set size to validation set size

In summary, this model clearly learns more than the *h1* model, but is still restricted to guesswork for some samples. It is likely that it learned gender, but in addition uses the second neuron to remember one of the three antecedent candidates, which sometimes proves to be useful.

Three hidden neurons (Figure 5.4) are the first to achieve perfect performance for a 9/1 and even a 1/1 ratio (Figure 5.4a, Figure 5.4c).

A split of 3/1 (Figure 5.4b) comes close, but is less monotonous and far more rocky then the next smaller split. An unlucky sample selection is less likely due to the relatively high split ratio, so we believe that with more training data there is generally also more confusion.



(a) $r = 9/1$

(b) $r = 3/1$

(c) $r = 1/1$

(d) $r = 1/3$

Figure 5.5: Accuracies for LSTM size 4 across different split ratios r of training set size to validation set size

The smallest split ratio of 1/3 (Figure 5.4d) performs as expected, i.e. with worse performance, slower and unstable generalization. However generalization still works well.

Four hidden neurons (Figure 5.5) are in general just faster and smoother than networks with three neurons. Generalization at the lowest ratio Figure 5.5d remains lower than in smaller models, but this could very likely be due to just an unfortunate selection of samples.

In summary, all models perform well above baseline, but likely due to different reasons. Fewer neurons seem to limit themselves to rough heuristics, which are hard to learn with smaller data ratios.

However, smaller ratios do not necessarily mean worse generalization, as can be seen by the consistent neck-to-neck accuracies of training and validation set. Since each sample is unique across the whole data, it is evident that the network learned abstract rules which it could apply to entities with similar usage patterns, e.g. the members of the masculine group. It is likely that the network learns to utilize cross-references and encodes them in a distributed fashion.

Unsurprisingly, higher split ratios cannot compensate for a small hidden size. If there is little to learn with, then there is little to learn from.

On the other hand, more hidden neurons can indeed make up for little data. The only exception so far is the devastating performance drop at the 1/1 ratio in Figure 5.3c compared to the smaller 1/3 ratio. A possible explanation is that as the network tries to figure out which feature to bet on with its few neurons, adding the wrong data just leads to confusion.

5.6 Neural Analysis

This section concludes the toy experiments of the previous sections with a first visual analysis of single neuron states, retrieved from the layers in selected samples of the test set.

The *Torch* code for reproducing the results in this chapter is available at:
https://github.com/kaumanns/DeepDiver

Section 5.6.1 introduces related work of plotting heatmaps over single neurons.

Section 5.6.2 introduces the terminology for describing the high-level behavior of LSTM gates in terms of their low-level activations.

Section 5.6.3 presents and interprets heatmaps over single neurons from different test set samples.

5.6.1 Related Work

Heatmaps are a method for intuitive visualization of low-level neural network activations. Their basic idea is to assign a color from a continuous spectrum, e.g. from blue to red, to a data point in order to make its relation to other data points more accessible to the human eye. Heatmaps have recently enjoyed rising popularity for shedding light on attentional hotspots in encoder-decoder machine translation systems (e.g. Hermann et al., 2015), as

well as for uncovering sudden excitements of single neurons in an LSTM language model (Karpathy et al., 2015).

The group notes that though heatmaps help gaining insights from gate activations, they are not guaranteed to uncover meaningful patterns when applied to distributed representations, such as memory layers. For this reason, interesting insights over the behavior of single LSTM memory cells via heatmaps has so far been restricted to a few lucky picks. Karpathy et al. (2015) found that most memory cells in a character-level LM exhibit somewhat uninterpretable activations by themselves. A few were shown to implement clear switches for brackets, quotes, and even continuous values for increasing position and whitespace indentations. Interestingly enough, this property has also been confirmed for some biological neurons that exhibit a form of *sparse coding* of features (Quiroga et al., 2008).

These heatmaps prove that LSTM networks are able to put their gates to a good use and acquire and protect single pieces of valuable information, such as being inside a quote. For this reason, LSTM networks are probably a much better candidate for a heat-based visual analysis than other types of neural networks whose main units are constantly updated with new information, such as feedforward MLPs and RNNs based on memory compression (i.e. simple RNNs), and are thus encouraged to learn pure distributed representations.

5.6.2 States of Gates

This section introduces terms to describe the activity of a single LSTM gate.

LSTM gates are commonly activated by the logistic function (*sigmoid*). This puts their states along the following range:

- A *sigmoid*-activated gate neuron is open when its state is > 0.9.
- A *sigmoid*-activated gate neuron is closed when its state is < 0.1.

These descriptions are equivalent to the ones defined by Karpathy et al. (2015) who describe open gates as “left-saturated”, and closed gates as “right-saturated”.

However, these terms are merely shortcuts to describe the low-level activation of a single gate, not its actual meaning for the role of the gate within the network. We therefore propose a new pair of terms that describe a *policy* to which a gate in that state adheres.

5.6.2.1 Definition of Gate Policies

First-order neurons, such as LSTM *memory cells*, can be difficult to interpret because each *memory cell* can be part of a larger distributed representation. For LSTM gates,

this is not the case, as each gate is trained to control a single *memory cell*. This allows us to define meaningful descriptive terms (*policies*) for the behavior of single gates:

- ***conservative*** gates inhibit the flow of information from upstream to downstream (*input gate*, *output gate*) and protect or retain the current *memory cell* states (*forget gate*).
- ***progressive*** gates allow the current *memory cells* to change in the light of new information (*input gate*, *forget gate*) and let information flow further downstream (*output gate*).

The translations from states to policies for each gate type are summarized in Table 5.3.

State	Input gate	Forget gate	Output gate
Open	progressive	conservative	progressive
Closed	conservative	progressive	conservative

Table 5.3: Policies of gate types with respect to their states. Note that the policies of the *forget gate* are flipped compared to the other gates.

An important difference between the *output gate* and the *input gate*/*forget gate* is that the *output gate* does not control write-access to the *memory cell* layer, only read-access. One could therefore argue that a closed *output gate* does not really operate in a *conservative* fashion. However, a single closed *output gate* still affects the downstream layers by holding back information of its *memory cell*, thus in a way *conserving* the impact of neighboring *memory cells*.

5.6.2.2 Caveats

There are three caveats and gotchas attached to the policies of each gate. It is important to keep them in mind when interpreting the plots in the upcoming sections.

All gates look back

One could argue that an open gate also employs some *conservative* properties, thanks to the recurrent connections from the *block output*. These connections effectively make an open gate use past information. Since this is only indirectly *conservative* and applies to all gates equally, we choose to not let it count as *conservative* policy.

The *forget gate* policy is flipped

Note that the relation between state and policy of the *forget gate* is flipped compared to the other gates, i.e. open means *conservative*. This quirk is important to remember when interpreting the plots later.

In many-to-one models, the *output gate* policy depends on the time step

All gates affect the respective downstream layers, both recurrent and non-recurrent. In a many-to-one sequence classification model, this has two different meanings for the *output gate*:

1. At non-last time steps, the *output gate* affects recurrent connections (at $t+1$) and current downstream layers (at t), **but not** the current network output.
2. At the prediction step (i.e. the last step), the *output gate* affects only current downstream layers (at t), **including** the current network output.

We will see in Section 5.6.3 and Section 7.3 that this causes a significant change in the policy of the *output gates* at the last time step.

5.6.3 Heatmaps

Moreland (2009) proposes a range of continuous diverging color palettes for scientific visualizations. We adopt the cold-to-hot metaphor of a blue-white-red color map because it can be applied to both activation ranges in our LSTM network:

- From 0 (white) to 1 (red) for *sigmoid*-activated neurons (*input gate*, *forget gate*, *output gate*)
- From -1 (blue) via 0 (white) to 1 (red) for *tanh*-activated neurons (*block input*, *memory cell*)

These colors are use to generate heatmaps from the test set evaluations of a perfect model with 3 hidden neurons and a split ratio of 9/1. In particular, our goal is to find patterns that support our assumption that the network learned to represent the features that are necessary to solve this task: *animacy*, *gender*, and *order*.

Single neurons in a layer are denoted via superscript indices, e.g. *forget gate*^{3} for *forget gate* number 3.

Recall that the network does not know which of the three positioned inputs will turn out to be relevant to resolve the anaphor. Thus, the main question is whether the network solves the task by brute-force memorization of each of the 7 target tokens, or learns to model the target space via a combination of higher-level features.

5.6.3.1 Output Gates

An overview of the *output gate* behavior is shown in Figure 5.6 (*feminine* samples), Figure 5.7 (*masculine* samples), and Figure 5.8 (*neuter* samples) .

Overall, *output gates* exhibit a progressive tendency, i.e. are mostly open, with a few selected shutdowns around noise words.

Recall that during the sequence, *output gates* feed only the recurrent connections to the next time step. Thus, the noise-word-related shutdowns of *output gate*^{1} and *output gate*^{2} can only be used to inform posterior layers of noisy context.

The patterns of permeability are similar among the *genders*. There are few differences in the visibility of selected shutdowns of *output gate*^{1} during noise words (`and` and the adverb-verb part) and the *animals*. However, it is unlikely that these differences are related to the *genders* of the names, as these different components of the context are independent from each other. We assume they stem from a more elusive pattern processing, related to individual samples.

Most importantly, at the query token, the *output gates* open up completely (with the inexplicable exception of `John` in Figure 5.7e, Figure 5.7k). This is a strong indicator that the state of the memory cell layer is assumed to be a perfect representation of the context, without any noise that would have to be filtered out before carrying their activations to the output layer. It is possible that *output gates* are useless for many-to-one sequence classification tasks, at least with shallow RNNs that do not feed into downstream layers during the sequence.

5.6.3.2 Input Gates

An overview of the *input gate* behavior is shown in Figure 5.9 (*feminine* samples), Figure 5.10 (*masculine* samples), and Figure 5.11 (*neuter* samples) .

Overall, the *input gates* are fairly wild mix of closed, open, and semi-open states which do not seem to be consistent across the *genders*. There is however a weak pattern of similarity within samples related to the antecedents.

For example, `Frank` seems to consistently keep *input gate*^{1} closed for all future time steps (e.g. Figure 5.10a, Figure 5.10b, Figure 5.10f), as does `dog` (e.g. Figure 5.11d, Figure 5.11e, Figure 5.11f). `Mary` causes *input gate*^{2} to become open (e.g. Figure 5.9a, Figure 5.9b, Figure 5.9c).

This is surprising because we did not expect the *input gate* to agree in terms of the antecedents. Instead, we expected the *input gates* to learn to become completely closed for specific input tokens, namely `and`, `<eos>`, `their`, and the adverb clause and verbs, similar to what we observe in Figure 5.11g and Figure 5.11d. As far as the network is concerned, these words are just noise and irrelevant for the task. However, we found that only *input gate*^{1} tended toward this expected conservative behavior, at least for `and`, the adverbs and verbs, though not in all samples.

(a) Mary, 1st *human*

(b) Mary, 1st *human*

(c) Mary, 1st *human*

(d) Susie, 1st *human*

(e) Susie, 2nd *human*

(f) Susie, 2nd *human*

(g) Susie, 2nd *human*

(h) Mary, 2nd *human*

(i) Mary, 2nd *feminine*

(j) Mary, 2nd *feminine*

Figure 5.6: Heatmaps of ***output gate*** states of ***feminine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. Note that in a sequence classifier, *output gates* feed only the recurrent connections, except for the last step where they carry the memory cell activations to the prediction layer. The patterns of permeability are mostly similar to those for the other *genders*. For selected noise words, as well as around the *animals*, *output gate*[1] seems to selectively shut down completely, possibly in order to communicate information about their relevance to posterior layers.

(a) Frank, 1st *human*

(b) Frank, 1st *human*

(c) Peter, 1st *human*

(d) Peter, 1st *human*

(e) John, 2nd *human*

(f) Frank, 2nd *human*

(g) Peter, 2nd *human*

(h) Peter, 2nd *human*

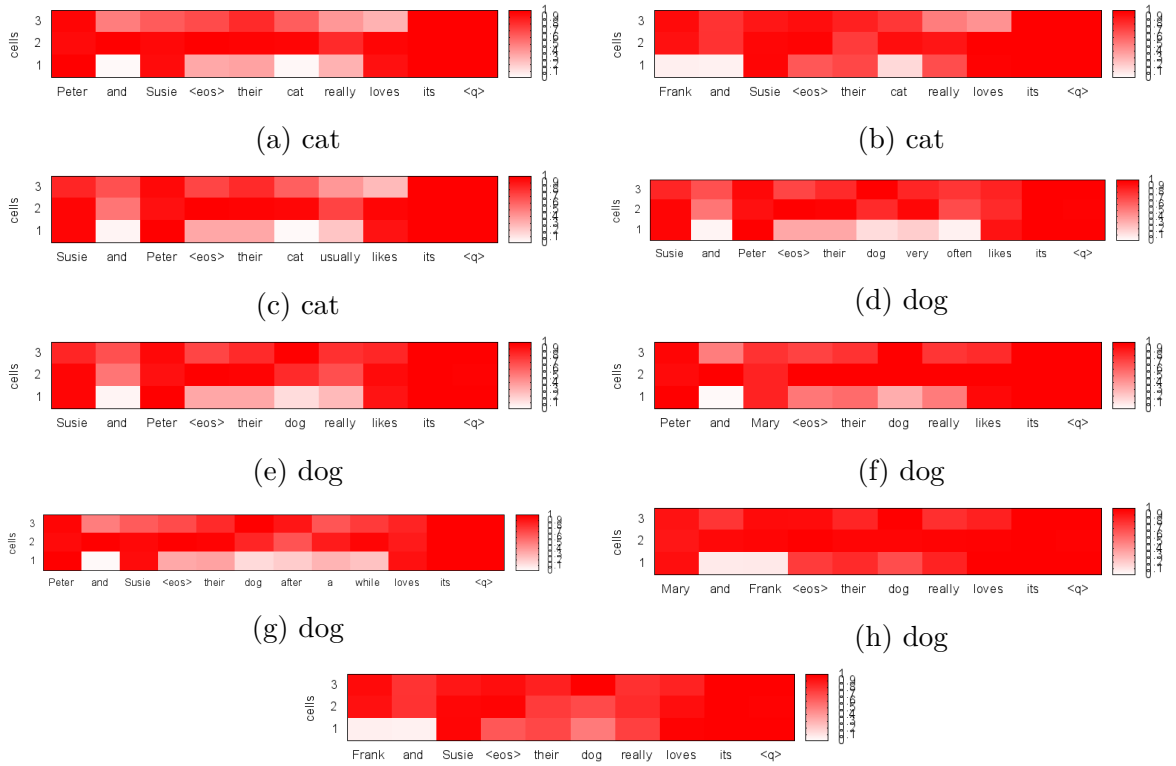
(i) Peter, 2nd *human*

(j) Peter, last *masculine*

(k) John, last *masculine*

(l) Frank, last *masculine*

Figure 5.7: Heatmaps of ***output gate*** states of ***masculine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. Note that in a sequence classifier, *output gates* feed only the recurrent connections, except for the last step where they carry the memory cell activations to the prediction layer. The patterns of permeability are mostly similar to those for the other *genders*. For selected noise words, as well as around the *animals*, *output gate*^{1} seems to selectively shut down completely, possibly in order to communicate information about their relevance to posterior layers. In samples related to *John* (Figure 5.7e, Figure 5.7k), *output gate*^{2} mysteriously shuts down from the *animal* on.



(a) cat (b) cat

(c) cat (d) dog

(e) dog (f) dog

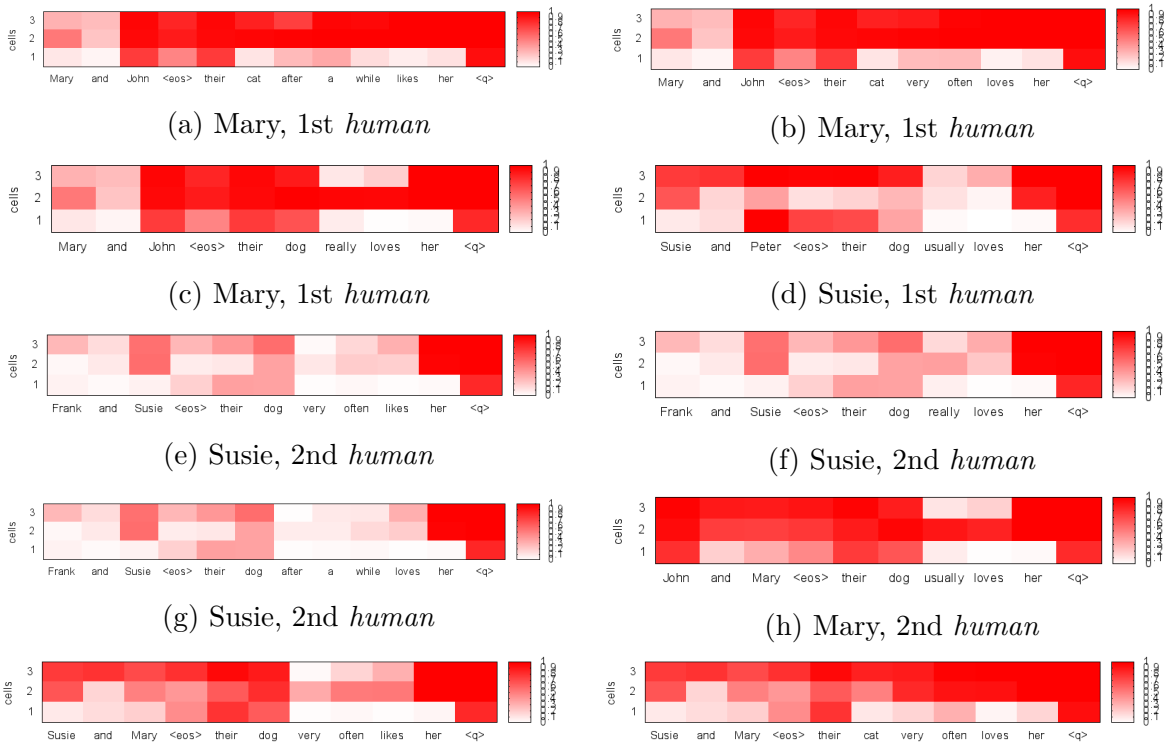
(g) dog (h) dog

(i) dog

Figure 5.8: Heatmaps of ***output gate*** states of ***neuter*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. Note that in a sequence classifier, *output gates* feed only the recurrent connections, except for the last step where they carry the memory cell activations to the prediction layer. The patterns of permeability are mostly similar to those for the other *genders*. For selected noise words, as well as around the *animals*, *output gate*[1] seems to selectively shut down completely, possibly in order to communicate information about their relevance to posterior layers.

This kind of processing does not seem to be optimal behavior for this task. We suspect that the *input gate* is spoiled with distracting influences besides the irrelevant inputs: the recurrent connections from the previous blockoutput and via the peepholes. Another explanation would be that, despite the already optimal performance, the network would need more training epochs to completely converge to an optimal policy for single gates. An indicator for a lack of training could be that *input gate*^{1} is still mostly open for `<eos>` and `their`, which seems to be a legacy state from the previous name token.

It would be interesting to see if we could boost performance (or the convergence rate in this case) just by leaving out these connections, given that we know that the input contains a high degree of noise inputs.



(a) Mary, 1st *human*

(b) Mary, 1st *human*

(c) Mary, 1st *human*

(d) Susie, 1st *human*

(e) Susie, 2nd *human*

(f) Susie, 2nd *human*

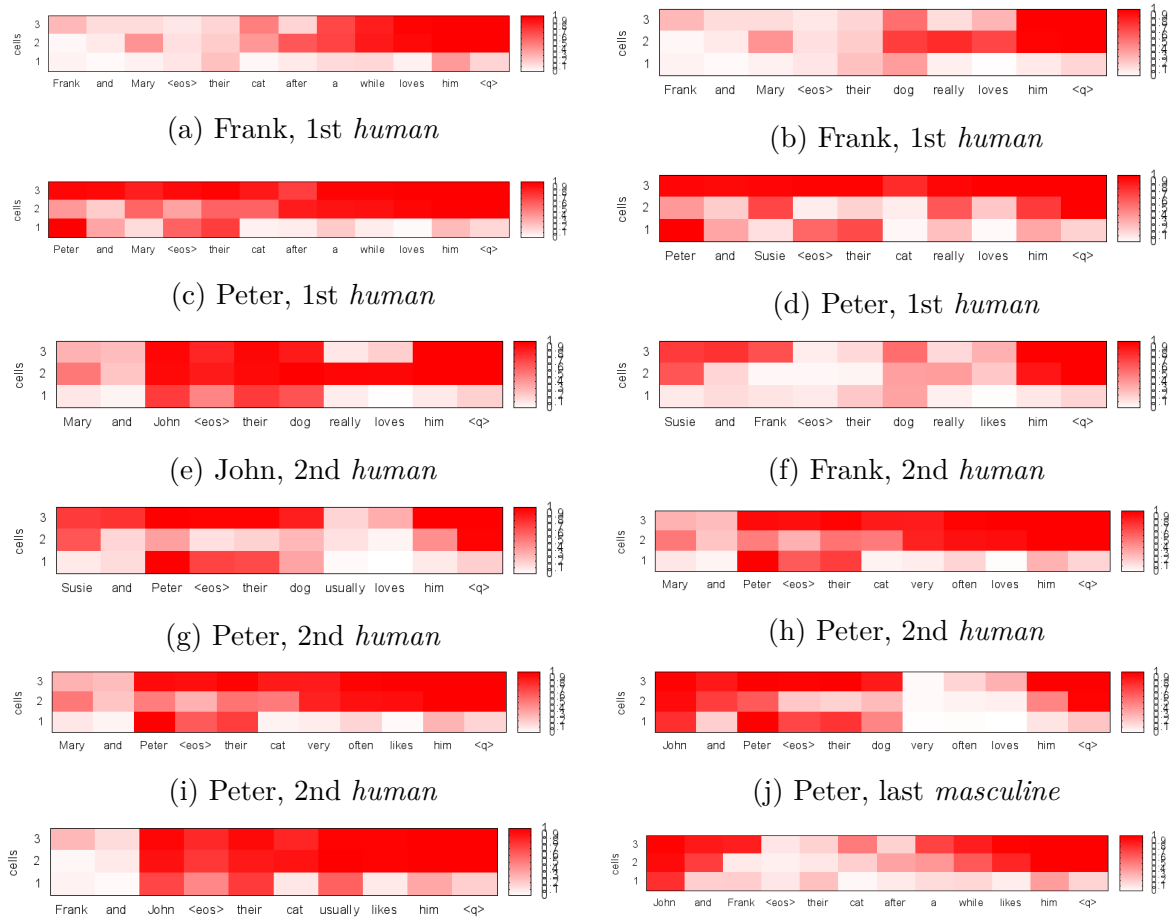
(g) Susie, 2nd *human*

(h) Mary, 2nd *human*

(i) Mary, 2nd *feminine*

(j) Mary, 2nd *feminine*

Figure 5.9: Heatmaps of ***input gate*** states of ***feminine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. It seems that activation patterns are shared among instances of the same name, which we find mysterious. The expected behavior for the *input gates*, namely shutting out all noise words, does not seem to be implemented. Only *input gate*^{1} tends toward this direction of policy.



(a) Frank, 1st *human*

(b) Frank, 1st *human*

(c) Peter, 1st *human*

(d) Peter, 1st *human*

(e) John, 2nd *human*

(f) Frank, 2nd *human*

(g) Peter, 2nd *human*

(h) Peter, 2nd *human*

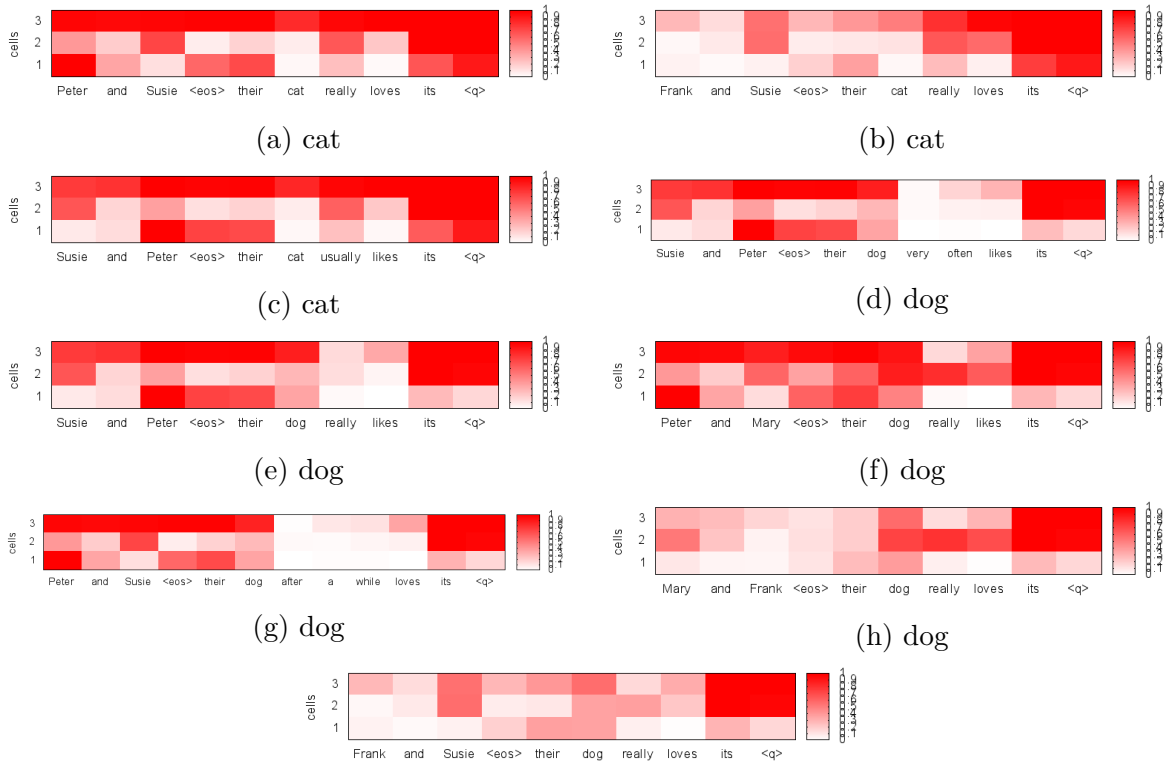
(i) Peter, 2nd *human*

(j) Peter, last *masculine*

(k) John, last *masculine*

(l) Frank, last *masculine*

Figure 5.10: Heatmaps of ***input gate*** states of ***masculine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. It seems that activation patterns are shared among instances of the same name, which we find mysterious. The expected behavior for the *input gates*, namely shutting out all noise words, does not seem to be implemented. Only *input gate*[1] tends toward this direction of policy.



(a) cat (b) cat

(c) cat (d) dog

(e) dog (f) dog

(g) dog (h) dog

(i) dog

Figure 5.11: Heatmaps of ***input gate*** states of ***neuter*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. It seems that activation patterns are shared among instances of the same name, which we find mysterious. The expected behavior for the *input gates*, namely shutting out all noise words, does not seem to be implemented. Only *input gate*^{1} tends toward this direction of policy. Only the adverb-verb clauses in Figure 5.11g and Figure 5.11d trigger this expected behavior.

5.6.3.3 Forget Gates and Memory Cells

An overview of the *forget gate* behavior is shown in Figure 5.12 (*feminine* samples), Figure 5.13 (*masculine* samples), and Figure 5.14 (*neuter* samples) .

Forget gates play a crucial role in the interpretation of the *memory cells*. They show exactly which *memory cells* are retained from the past (or not) and which *memory cells* are allowed to be overwritten by the current input, or merged with it. Therefore, they provide strong hints to the meaning of individual *memory cells*.

In general, the *forget gates* are very conservative (i.e. open), retaining most of the past *memory cell* states. Though they seem to assume the function of binary switches, it is not entirely clear which exact feature is controlled by each single *forget gate*, or even by the combination of all three *forget gates*. For example, in Figure 5.13, *forget gate*^{1} closes down after *Frank* and *John*, but not *Peter*, ruling out the possibility of a clean *masculine* switch.

An exception from this rule is the behavior of the *forget gates*at the anaphora step. Here, they distinctly encode the *gender* feature by the combination of their three states.

Animacy and Gender at Penultimate Steps

After the anaphor at penultimate steps, the *forget gates* suddenly exhibit a strong pattern for *gender*, probably triggered by the incoming connection from the input:

- After the *feminine* anaphor, only *memory cell*^{2} and *memory cell*^{3} are retained (Figure 5.15).
- After the *masculine* anaphor, only *memory cell*^{1} and *memory cell*^{2} are retained (Figure 5.16).
- After the *neuter* anaphor, only *memory cell*^{3} is retained (Figure 5.17).

It seems that the *forget gates* aggressively rule out a subset of prediction options by deleting the *memory cell* that represents this subset.

We assume that the model is most efficient, so we expect it to learn to retain a clean representation of the target token in memory, and therefore delete the neurons responsible for the target tokens to be ruled out.

Neuter

We check this behavior first by inspecting the activations after the *neuter* anaphor. This is the most simple to resolve for the model since it requires only two features: *animacy* and *animal*.

Here, only *memory cell*^{1} is retained, which hints to it being responsible for the *animal* feature in absence of *memory cell*^{2} and *memory cell*^{3}.

memory cell^1 assumes a consistent value of $\approx$ 1 for `dog` and a consistent value of $\approx$ 0.6 for `cat`. This small difference is then translated to two distinct *memory cell* patterns for both *animals* at the next time step.

We conclude that the network learned to separate *animals* from *humans* via the most simple representation possible: two values of one *memory cell*.

Human and genders

As the resolution between 5 possible *humans* requires both two more *genders* and sometimes the *order*, their inspection is more difficult.

For each *gender*, the *forget gates* retain two *memory cells*:

- *feminine*: *memory cell*^{2} and *memory cell*^{1}
- *masculine*: *memory cell*^{2} and *memory cell*^{3}

Note that *memory cell*^{1} is not universally discarded, rejecting the idea that it serves as an independent indicator for *animal*/*neuter*. It seems that the network learned to represent the two human *genders* via two distributed activations.

Since their only common *memory cell* is *memory cell*^{2}, which also separates it from the *neuter* samples, it is possible that its sole presence cues the network to assume a *human* context.

In all *human* cases, and analogously to the *animal* samples, the specific values of the *memory cells* from the penultimate time step are then translated to distinct patterns in the last time step that unambiguously describe each value of the *human* feature. These patterns are consistent for each name, independent from their order (i.e. first or second name mentioned).

Names during the sequence

The following patterns might only occur in this specific toy task, though they are interesting enough to be worth being mentioned.

Some names seem to consistently trigger a specific state in a specific *memory cell*. For example:

- `Peter` triggers a persistent positive excitement in *memory cell*^{1} (e.g. Figure 5.16c).
- `John` triggers a persistent negative excitement in *memory cell*^{1} which is canceled out when a same-*gender* antecedent follows (Figure 5.16j, Figure 5.16l). See also Section 5.6.3.3 for a discussion of the *order* feature.
- `Mary` triggers a mostly negative excitement in *memory cell*^{2} (e.g. Figure 5.15j).

These observations seem to contradict the idea from the previous section that the network learned a high-level *gender* feature, and instead just finds a way to represent each target individually.

A special case is `Susie` which is much less directly traceable. It seems that the weakly positive excitement it triggers in *memory cell*^{2} from the second position jumps to a negative excitement in *memory cell*^{3} as soon as the *animal* candidate comes in (e.g. Figure 5.15e). A similar effect can be observed in Figure 5.17a, Figure 5.17g, and Figure 5.17i.

If this interpretation is correct, it means that the network is able to translate the content from one *memory cell* to the other in order to make space for new incoming information. Further experiments would have to provide more evidence for this hypothesis.

(a) Mary, 1st *human*

(b) Mary, 1st *human*

(c) Mary, 1st *human*

(d) Susie, 1st *human*

(e) Susie, 2nd *human*

(f) Susie, 2nd *human*

(g) Susie, 2nd *human*

(h) Mary, 2nd *human*

(i) Mary, 2nd *feminine*

(j) Mary, 2nd *feminine*

Figure 5.12: Heatmaps of ***forget gate*** states of ***feminine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. Overall, *forget gates* are consistently conservative (open) with a few highly focused closed states which trigger forgetting specific *memory cells*, possibly to rule out alternatives. Patterns are related to antecedents and anaphora only. *forget gate*^{1} is closed after all *feminine* anaphora. There is no specific permeability after *feminine* antecedents. *forget gate*^{1} is closed after all *masculine* names.

(a) Frank, 1st *human*

(b) Frank, 1st *human*

(c) Peter, 1st *human*

(d) Peter, 1st *human*

(e) John, 2nd *human*

(f) Frank, 2nd *human*

(g) Peter, 2nd *human*

(h) Peter, 2nd *human*

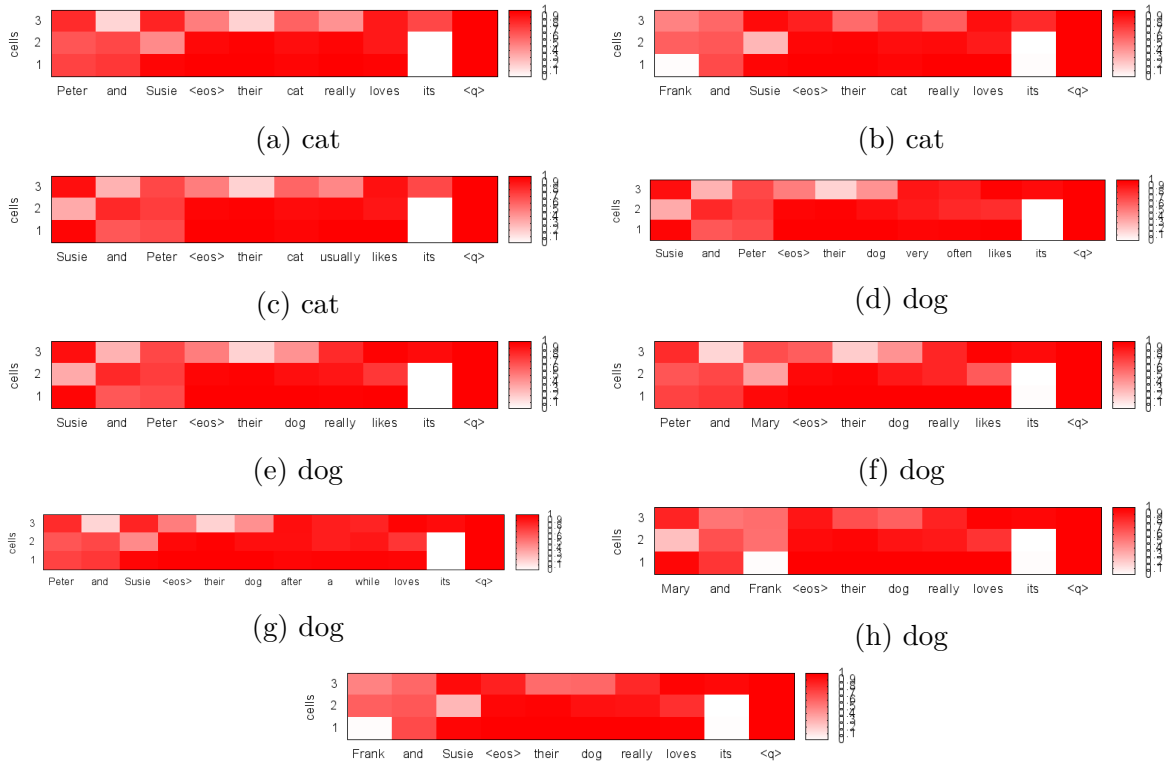
(i) Peter, 2nd *human*

(j) Peter, last *masculine*

(k) John, last *masculine*

(l) Frank, last *masculine*

Figure 5.13: Heatmaps of ***forget gate*** states of ***masculine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. Overall, *forget gates* are consistently conservative (open) with a few highly focused closed states which trigger forgetting specific *memory cells*, possibly to rule out alternatives. Patterns are related to antecedents and anaphora only. *forget gate*^{3} is closed after all *masculine* anaphora. *forget gate*^{1} is closed after *masculine* antecedents *Frank* and *John*.



(a) cat (b) cat

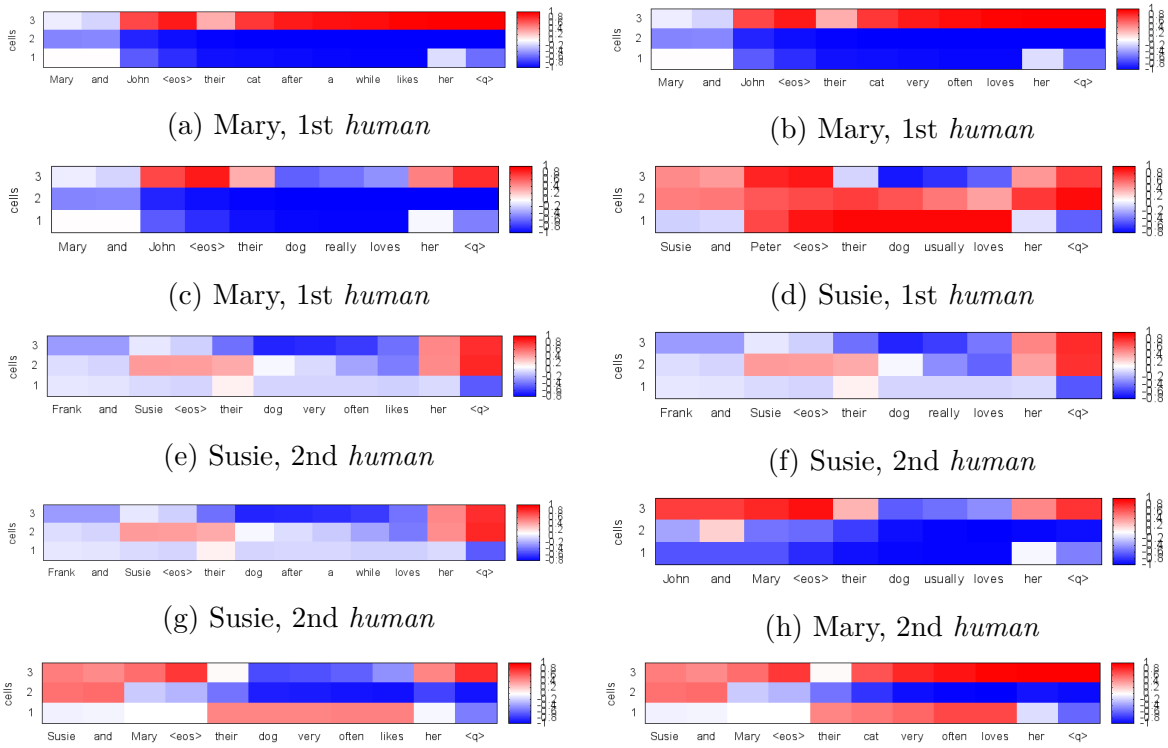
(c) cat (d) dog

(e) dog (f) dog

(g) dog (h) dog

(i) dog

Figure 5.14: Heatmaps of ***forget gate*** states of ***neuter*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. Overall, *forget gates* are consistently conservative (open) with a few highly focused closed states which trigger forgetting specific *memory cells*, possibly to rule out alternatives. Patterns are related to antecedents and anaphora only. *forget gate*^{1} and *forget gate*^{2} are closed after all *neuter* anaphora. There is no specific permeability after *neuter* antecedents.



(a) Mary, 1st *human*

(b) Mary, 1st *human*

(c) Mary, 1st *human*

(d) Susie, 1st *human*

(e) Susie, 2nd *human*

(f) Susie, 2nd *human*

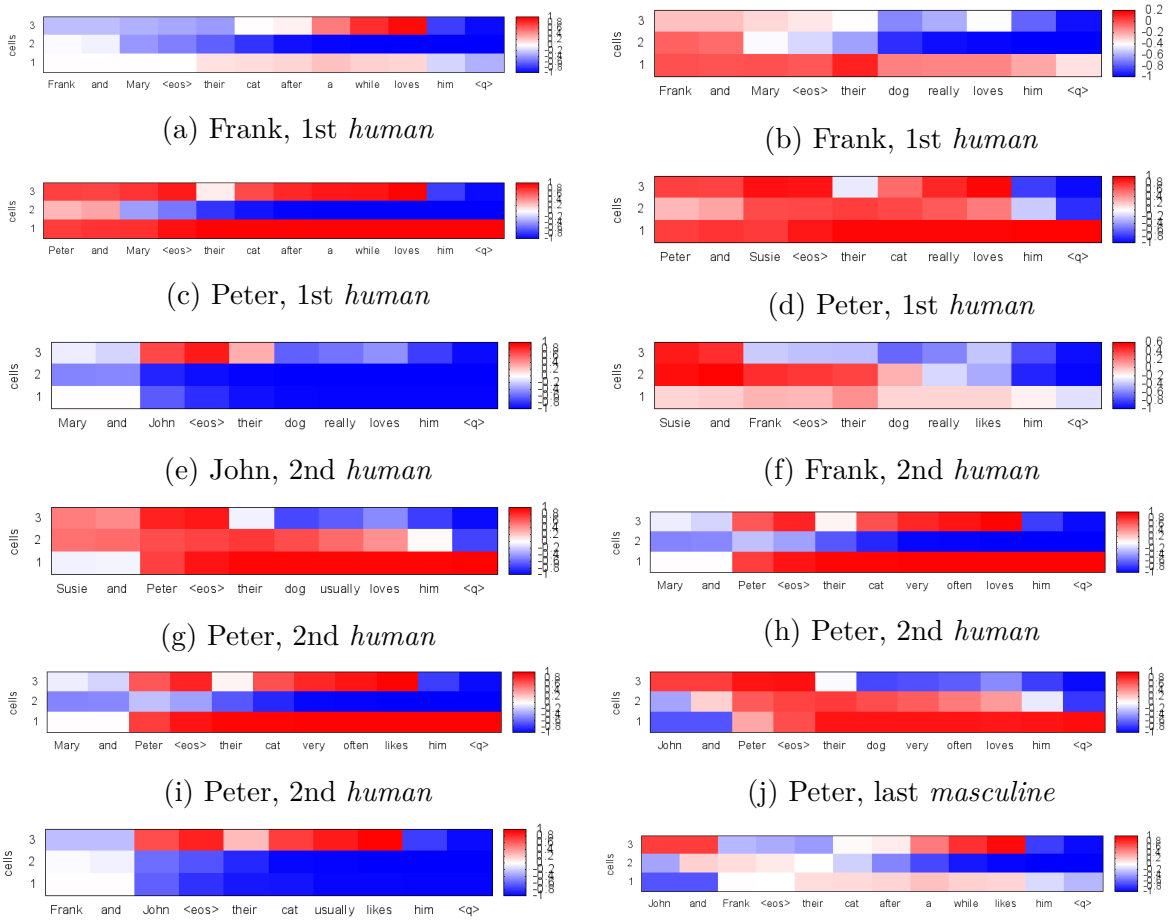
(g) Susie, 2nd *human*

(h) Mary, 2nd *human*

(i) Mary, 2nd *feminine*

(j) Mary, 2nd *feminine*

Figure 5.15: Heatmaps of ***memory cell*** states of ***feminine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. For example, all penultimate *memory cell*^{3} for *feminine* targets are positively excited. *Mary* triggers a mostly negative excitement in *memory cell*^{2} (e.g. Figure 5.15j). *Susie* is less directly traceable. It seems that the weakly positive excitement it triggers in *memory cell*^{2} from the second position jumps to a negative excitement in *memory cell*^{3} as soon as the *animal* candidate comes in (e.g. Figure 5.15e). A similar effect can be observed in Figure 5.17a, Figure 5.17g, and Figure 5.17i.



(a) Frank, 1st *human*

(b) Frank, 1st *human*

(c) Peter, 1st *human*

(d) Peter, 1st *human*

(e) John, 2nd *human*

(f) Frank, 2nd *human*

(g) Peter, 2nd *human*

(h) Peter, 2nd *human*

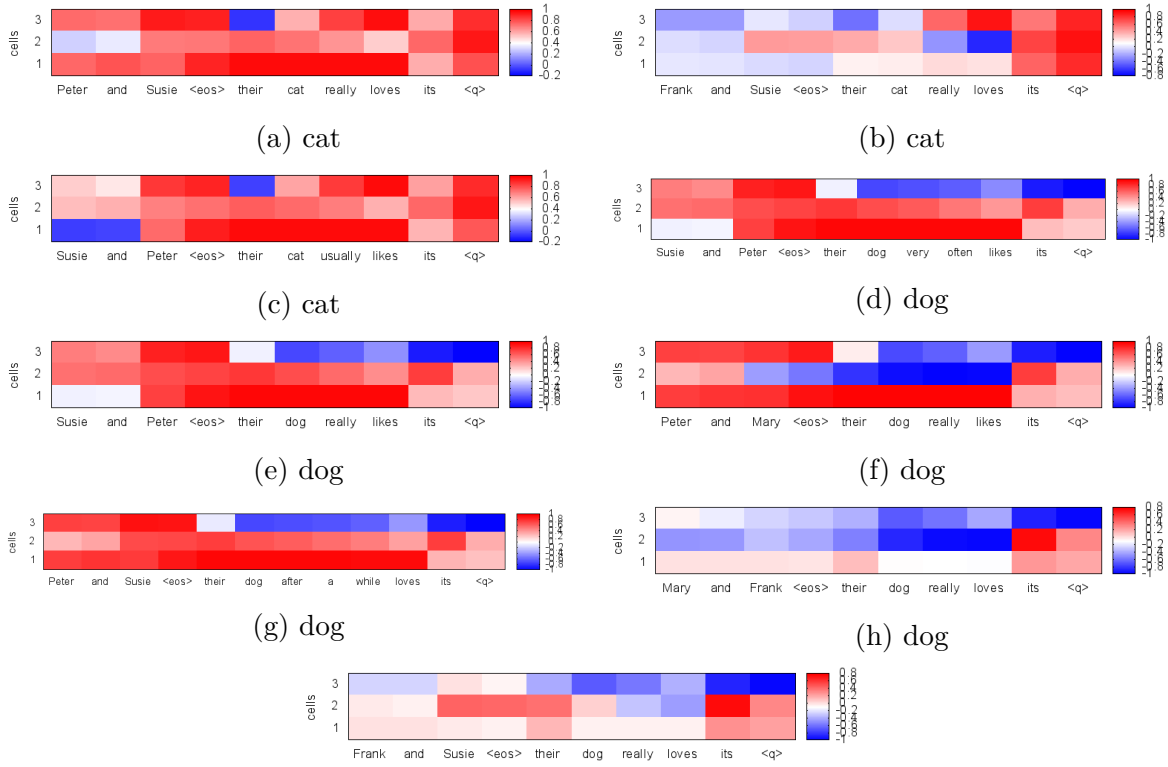
(i) Peter, 2nd *human*

(j) Peter, last *masculine*

(k) John, last *masculine*

(l) Frank, last *masculine*

Figure 5.16: Heatmaps of ***memory cell*** states of ***masculine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. For example, all penultimate *memory cell*^{3} for *masculine* targets are negatively excited. *Peter* triggers a persistent positive excitement in *memory cell*^{1} (e.g. Figure 5.16c). *John* triggers a persistent negative excitement in *memory cell*^{1} which is canceled out when a same-*gender* antecedent follows (Figure 5.16j, Figure 5.16l).



(a) cat (b) cat

(c) cat (d) dog

(e) dog (f) dog

(g) dog (h) dog

(i) dog

Figure 5.17: Heatmaps of ***memory cell*** states of ***neuter*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. For example, all penultimate *memory cell*^{2} for *neuter* targets are positively excited. A positive penultimate *memory cell*^{3} is related to the *cat* target. A negative penultimate *memory cell*^{3} is related to the *dog* target.

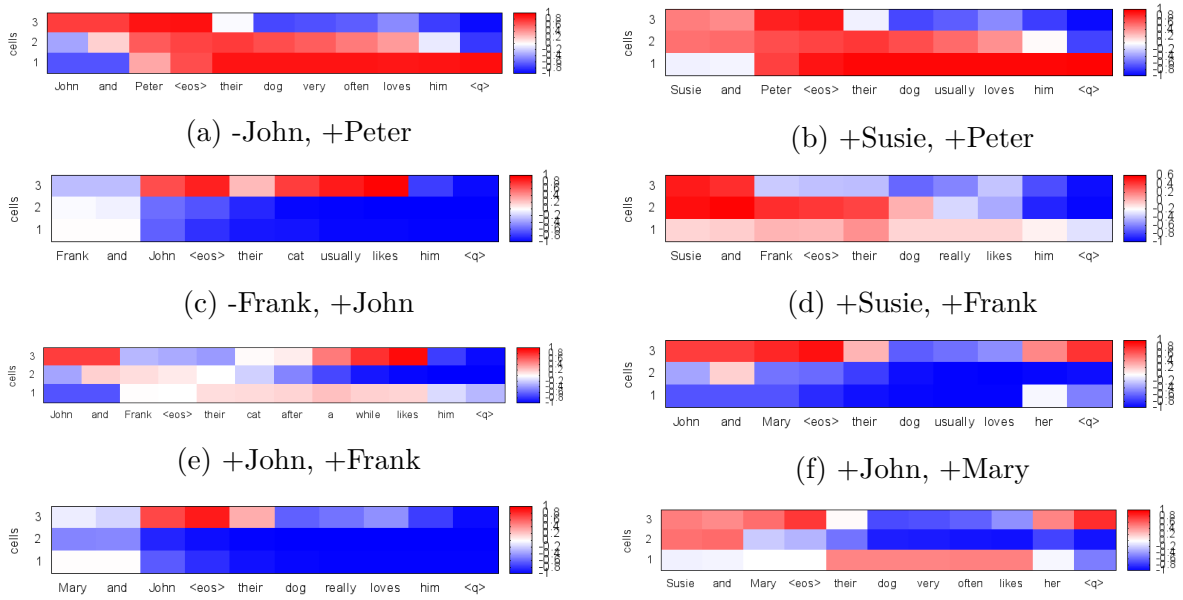
Order

The remaining question is how the networks knows to predict the correct *human* (i.e. the last one) if both names have the same *gender*. For this, we compare samples that feature the same names at different positions (Figure 5.18).

It seems that the *memory cell* activations are quite similar after the second name, both for samples with two mixed and two same *genders*.

Moreover, the activation patterns triggered by the same name at different positions are largely equivalent (e.g. `John` in Figure 5.18c), except when the sample features two names of the same *gender* (e.g. in Figure 5.18a). In the latter case, the activations are abruptly replaced when the second same-*gender* name comes in.

It thus seems obvious that the network learned to overwrite the *memory cells* if it detects a second name of the same class (i.e. *gender*) as the previous name, instead of choosing the more costly option to represent an *order* feature within the memory layer.



(a) -John, +Peter

(b) +Susie, +Peter

(c) -Frank, +John

(d) +Susie, +Frank

(e) +John, +Frank

(f) +John, +Mary

(g) +Mary, +John

(h) -Susie, +Mary

Figure 5.18: Heatmaps of *memory cell* states in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. This figure contains the subset of samples that shows that the network does not learn an explicit *order* feature in order to resolve same-*gender* samples. Rather, it learns to overwrite its *memory cells* when it receives a second instance of the same *gender*. The labels indicate which names are preserved in the subsequent *memory cell* patterns ("+" for preserved, "-" for canceled out). Note how in same-*gender* constellations the second name completely cancels out the first name.

5.6.3.4 Summary

We found strong indicators that the network indeed learns to handle the three features required for this task. It seems that the human *genders* are each represented by the mere presence of single *memory cells*, i.e. *memory cells* that are not close to 0. Further subclassing into the specific predictions seems to happen via an additional time step (the query token), during which small activation differences in these remaining "trigger *memory cells*" are translated into a full memory layer representation of the target.

We found that not all features are represented via the *memory cells*. The *order* feature seems to be handled via a force overwrite at the second same-*gender* human.

The *output gate* exhibited a mostly *progressive* policy and thus seemed to be relatively useless for this shallow RNNsequence classifier, even after the query token. The fact that all *output gates* were open even after the query token can be explained by the observation that the model resolved the human *gender* feature at the anaphor, and then had still one more step to condense this *gender* representation in the memory layer down to a single antecedent. It is therefore likely that the omission of the query token would force the *output gate* to assume a more selective role in order to map the *gender* representation directly to the prediction of the proper antecedent.

Despite the low-complexity setup of the experiments, we are not able to confidently verify that the network indeed resorts to higher-level features (e.g. *gender*). Instead, each of the five antecedent candidates is encoded by a specific imprint in the memory layer which is eventually mapped to the correct prediction. It is possible that the ratio of data complexity to network complexity is too low, i.e. even a tiny network is still powerful enough to brute-force remember the three antecedents and eventually pick the correct one.

Further experiments would thus have to increase this ratio in order to force a weak network to learn higher-level features. Eventually, this toy task approach will have to face the question if the insights can be extrapolated to larger networks and more complex data.

The following chapters will try to tackle this question and explore more complex setups on real-life corpora.

5.7 Conclusion

This chapter applied a Long Short-Term Memory (LSTM) network to the task of antecedent classification in order to solve the problem of Anaphora Resolution (AR). It was shown how restricted toy setups and low-level analysis provide interesting insights into the dynamics of a Recurrent Neural Network (RNN) and how it deals with little memory

and unfavorable set splits. The heatmaps presented in the last section gave a first glimpse into what makes an RNN tick. Further experiments would have to verify and extend our findings.

More complex data

Further work in this direction should iteratively add features and complexity to the toy data set. For example, this dataset could be extended by having not only humans compete for the same positions, but all nouns. We could then add more grammatical variation and noise that makes learning a bit harder, including noise after the anaphor. For example:

```
John and the cat. It really likes his car <q>
John and a dog. He really likes its magnificent hair <q>
John and a dog. She really likes their attitude and spirit <q>
```

These new sentences would have to be carefully designed such that features that are relevant to the AR task can still be defined apriori.

More complex representations

This setup will inevitably hit a wall, when the features necessary to solve the task cannot be derived from the syntax alone. This is especially important for synthetic and agglutinative languages. More complex word representations, such as morphosyntactic features, word embeddings, or convolutional representations of characters, would then be necessary.

More complex tasks

Furthermore, new tasks that require natural language understanding can be cast into this framework of question-answering and sequence classification. An obvious next step is coreference resolution with true symmetric mentions. But also more complex reasoning tasks, such as described by Weston et al. (2014), could be tried against this framework.

Real-life domains

Naturally, the goal should be the application of real-life data to this framework.

Latent features

So far we have focused on encouraging the network to learn well-defined linguistic features, or at least to emulate a good guess for them.

Both more complex tasks and real-life data however will likely introduce latent features which are harder to grasp via the low-level neuron-wise analysis we have described in this chapter. New methodologies for visualizing latent features are therefore necessary.

More complex network

It is likely that a simple LSTM would not suffice for these enhanced problems, especially when the sequences become longer and the network has a hard time remembering all information, just in case it becomes relevant. Bidirectionality seems like a mandatory extension.

Part of these extensions will be introduced in the next chapter.

Chapter 6

Mention Pair Identification

This chapter extrapolates the methodology from Section 5 from simple synthetic datasets to real-life natural language data. Instead of a small synthetic corpus, this system is trained on a subset of the CoNLL-2012 data set for Coreference Resolution (CR) with annotated chains of coreferent mentions. In order to keep this problem feasible with a simple architecture, we reallocate the complexity mass of the task. Instead of asking a complex question (*which word*) on top of simple synthetic data, we ask a simple binary question (*coreferent or not*) on top of complex real-life data. We will show that a generic deep bidirectional LSTM networks can identify mention-pairs well above baseline, even if only sparse word encodings are presented.

Section 6.1 provides an overview of related work regarding machine-learning based mention-pair identification.

Section 6.2 describes the data pipeline required to make the raw CoNLL-2012 data fit for the task.

Section 6.3 discusses the architectural variants and the training regime for the experiments.

Section 6.4 reviews a high-level performance analysis of the different architectures.

Section 6.5 investigates the performance of the best model on various subsets, specified by linguistic classes of mentions.

Section 6.6 summarizes the results and proposes further experiments.

6.1 Related Work

Stuckardt (2007) claim to be the first to suggest the application of neural networks to the task of Anaphora Resolution (AR). Their system relies on a rich set of hand-engineered features and produces mixed results compared to an alternative based on decision trees. Most notably, they find that the neural network approach has difficulties resolving non-possessive relations and suggest further refinements of the feature set.

Khashabi (2013) introduces recursive neural networks for complex linguistic tasks, such as relation extraction and entity recognition. They state that Coreference Resolution (CR) is "interlocked" with these problems, since CR aims to link mentions that point to the same object. Both tasks require a semantic understanding of the context in order to resolve the complex ways in which linguistic objects are related.

Recently, Memory Networks have successfully been applied to a range of varied Question-Answering (QA) tasks. Kumar et al. (2015) show that the problem of CR, as a subtask of QA, is implicitly performed when such as system is trained.

Cheng and Voigt (2015) propose an end-to-end system for full CR that is based on an LSTM language model and uses few external features. Tokens are encoded via dense embeddings (50-dimensional GloVe vectors). Though they claim that they use "virtually no hand-engineering of features", they enrich the token representations by features for end-of-sentence, capitalization, as well as third-party labeling of parts-of-speech and speaker IDs, both given by the CoNLL-2012 dataset. The word representations of each document are fed into an LSTM language model to generate a sequence of word-level outputs. Mentions are represented by an average of their respective word embeddings. The output of the model is a correlation matrix denoting the coreferring mentions, trained against the true correlations. This sophisticated network design has the advantage of providing end-to-end coreference resolution, including the initial mention detection. However the system performs only on the lower-end spectrum among state-of-the-art systems.

The model described by Cheng and Voigt (2015) provides evidence for the general applicability of a minimal-feature CR system and comes very close to our implementation, with a couple of additions on top. The overall design of the network is not a generic Recurrent Neural Network (RNN), but features several additions that fine-tune its design to solve the specific problem of CR. Furthermore, it is not completely devoid of features provided by external tools, for example part-of-speech, speaker ID, and pre-trained word embeddings.

Clark (2015) train deep neural networks for binary mention-pair classification. The network is presented with both mentions, each represented by a set of features that captures internal as well as contextual properties, e.g. word embeddings for the first, last, and head words, the average embedding of all words, the average embedding of certain position-

dependent context words, as well as the distance between the mentions and their degree of lexical matching. The word embeddings are 50-dimensional vectors, trained with *word2vec* on Wikipedia and Gigaword. The concatenation of all these features is fed into a two-level network with rectified linear units (ReLU), optimized against the negative log-likelihood of the target. This setup performs on par or better than many state-of-the-art systems while using a much smaller number of handcrafted features. The authors attribute this boost to more efficient learning and better generalization due to the fact that similar words are assigned similar representations. Though coming close, this system admittedly falls short of providing a true end-to-end solution and thus its performance still depends on a number of external factors, beside the corpus itself.

Zhekova and Kübler (2013) introduce a binary classifier for mention head detection, a preliminary subtask of CR for finding the heads of candidate mentions in a text. They implement a k-nearest neighbor decision model that decides on a per-word basis whether that word is the head of a mention or not. Since their system aims to be multilingual, they define a set of 14 language independent features that mostly make use of part-of-speech information of mention tokens at selected positions. This information is extracted from the pre-annotated CoNLL-2012 corpus. Their system outperforms heuristic-based competitors and compares favorably to rule-based approaches, close to the gold annotation, thereby showing that generic machine learning on simple features can go a long way, especially when it comes to bootstrapping a multilingual system from very little data.

The systems described above make clear that coreference resolution, and even mention-pair identification, is still struggling with the tradeoff between complex rules and complex features, while the combination of both usually gives the best results. Contrary to these approaches, our goal is to truly strip away all external features in order to get an unobfuscated view on the network activations.

6.2 Data Preparation

This section describes the preprocessing pipeline from raw CoNLL-2012 data to network-ready samples of full discourses with annotated mention pairs.

See Section 9.2 in the appendix for examples from the resulting data set.

6.2.1 Overview

The challenge data of CoNLL-2012 provide some basic syntactic and semantic information: part-of-speech, parse tree, predicate lemma, word sense, speaker, named entities, and

predicate arguments (Pradhan et al., 2012).[1] The purpose of these pre-annotated features is to provide a common ground in the framework of a shared task, and to eliminate some variables (the performances of preprocessors) that would skew a fair CR-focused comparison between the participants.

Examples for sentences in CoNLL-2012 with parentheses around mentions:

Nested mentions

```
(My (life))
((The eighth Army) guerillas)
((The eighth Route) Army) and ((its) base areas)
(The monument) to ((the Hundred Regiments) Offensive).
```

Long mentions

```
(A closely connected transport network, with a road for
every village and defensive towers on every road)
```

Neural networks are known to be quite data-hungry for achieving good performances. Since CoNLL-2012 relies heavily on a rich set of manual and semi-automatic feature annotations, it naturally lacks the volume that is typically required for large-scale network applications.

Despite the given lack of data, we apply two heuristics to further reduce and simplify the set:

1. Removing the longest sequences until 90% of the original number of samples are left. This is necessary to keep the maximum sequence length in check.

2. Restricting the samples to a subset that satisfies a specific linguistic phenomenon: containing at least one mention with a reflexive pronoun.

The reason for these reductions is that our initial experiments on the full data set did not perform well enough to make us confident about the value of the subsequent analysis. The poor performance seemed to be a result of the lack of data variety, as well as the fact that though LSTM networks are able to look much further into the past, the vanishing gradient problem ultimately hits the three gates as well (see Section 3.2). It remains to be seen if the resulting lower volume of training data is sufficient for proper generalization over unseen test data.

All of the annotated features are discarded in our approach except the bare mention annotations. Avoiding using external features and reducing the complexity of the data is an approbated strategy for stripping away outside influences and focus on the expressiveness

[1] http://conll.cemantix.org/2012/data.html

of a particular architecture itself (Khashabi, 2013). The steps of the data pipeline are summarized in Figure 6.1.

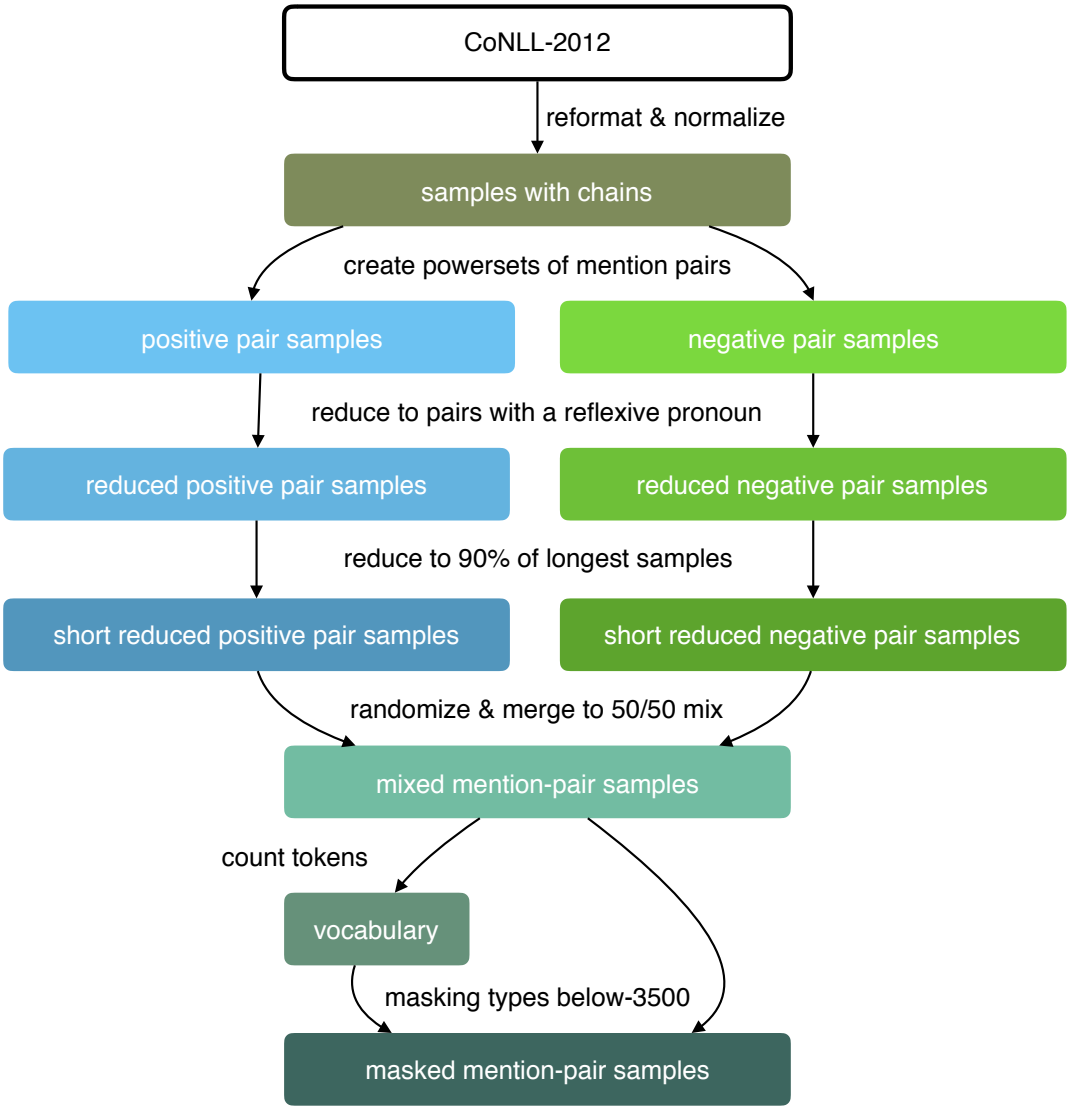


Figure 6.1: Preprocessing pipeline for CoNLL-2012, from raw data to samples with mixed positive and negative mention-pair annotations.

6.2.2 Reformat CoNLL-2012

Single words in CoNLL are assumed to always be separated by whitespaces. Each word in CoNLL is annotated with semantic and syntactic information. We discard all annotations

Orthographic class	Token
Ordinal numbers	<ord>
Time periods (e.g. ”1980s”)	<date>
Any other number (e.g. dates, times, cardinal)	<num>
Final punctuations	<eos>
Commata	<comma>
Quotations	<quot>
Other punctuations	<punct>
Interjections	<interj>
Any token that contains other non-word characters than hyphens or single apostrophes	<unk>

Table 6.1: Summary of orthographic normalizations for CoNLL-2012.

but the sentence boundaries and the coreferences.

Coreference annotations are reformatted to an XML-style annotation. In addition to common XML markers (such as `<eos>` for end-of-sentence), we introduce the following elements to annotate mentions of a chain that belong to the same referent N:

- `<cN>` : opening mention of chain that refers to referent N
- `</cN>` : closing mention of chain that refers to referent N

Example:

```
Congratulating <c1> herself </c1> on getting a low price <comma> and
confident that <c2> the market </c2> would rebound <comma> <c1> Mrs. Chang
</c1> happily put down the first payment and watched <c3> <c4> <c1> her </c1>
brother 's </c4> family </c3> move in <eos>
```

Note that we include all types of mentions, i.e. separate, nested, recursive, and overlapping mentions.

6.2.3 Normalization

Orthographic normalization is summarized in Table 6.1.

Word-surrounding markup, such as asterisks (∗), are assumed to be irrelevant and are thus removed. We do not apply lower-casing in order to help the network distinguish proper names from common nouns. Abbreviations and acronyms are kept intact, since they will be replaced by the special unknown token (`<unk>`) in later stages if they turn out to be too rare.

Set	Count
total contexts	84,796
... discarded singleton contexts	59,040 (69.63%)
... remaining contexts	25,756 (30.37%)

Table 6.2: Context counts

Linguistic normalizations are kept to a minimum. *n't* is normalized to *not*, else we keep contractions (*you 're*, *I 'll*, *he 's*) and hyphenated words as they are (i.e. as separate tokens).

6.2.4 Splitting CoNLL-2012 Into Sentences

CoNLL-2012 is provided as a set of discourses, each consisting of an ordered list of sentences. For this experiment, we define a *context* as a *single sentence* only, ignoring their respective embedding in the greater discourse, the reason being that sequences of 100 tokens and longer have proven to be very difficult to train.

Coreferences most often span several sentences within the discourse. Separating the samples by sentence boundaries thus creates a great number of orphaned mentions (singletons). Indeed, the vast majority of the resulting single-sentence contexts consist solely of singletons, e.g.:

```
<c1> They </c1> went by land to <c2> the same place
<c3> he </c3> went </c2> <eos>
```

We discard these contexts because they would not yield any positive samples. As a consequence, the number of usable contexts is reduced to 30.37% (Table 6.2).

Duplicate sentences are removed.

6.2.5 Creating Powersets of Mention Pairs

Each context now contains one or more mention chains. The following steps transform these chains into a labeled data set of positive and negative samples, each of which contains only two annotated mentions. As we transform the chain of each sentence into a set of all its positive and negative mention pairs, each sentence gives rise to an exponential number of derived labeled samples.

6.2.5.1 Definition of the Powersets

For the positive samples, each mention is combined with each other mention of the same chain.

For the negative samples, each mention is combined with each other mention of other chains.

Definitions:

- M^c : set of all mention sets (i.e. the set of all chains) in context c
- M_i^c : mention set i in context c

The number of positive samples $|Pos_c|$ and the number of negative samples $|Neg_c|$ of context c are then defined as follows, using the binominal coefficient $\binom{n}{k}$ to compute the number of ways to select k elements out of n, and the Kronecker-Delta annotation for conditional sums:

$$|Pos_c| = \sum_{i=1}^{|M^c|} \binom{|M_i^c|}{2} \tag{6.1}$$

$$|Neg_c| = \sum_{i=1}^{|M^c|} \sum_{j=1}^{|M^c|} |M_i^c|[i \neq j] \tag{6.2}$$

$$\equiv \binom{\sum_{i=1}^{|M^c|} |M_i^c|}{2} - |Pos_c| \tag{6.3}$$

Equation 6.2 is a simple nested loop over the mentions with a non-equality condition. It is equivalent to computing the powerset of mention pairs over all chains, and then subtracting the number of positive pairs (Equation 6.3).

Note that the first summand of Equation 6.3 is the same as Equation 6.1 if there is only one chain in the context, hence canceling out the second summand. In this case, no negative samples could be generated from the respective context c. In fact, this happens quite often and eventually leads to a negative sample set that is much smaller than the positive one (see Section 6.2.7).

6.2.5.2 Creating the Powersets

Each chain is exploded into all possible positive mention pairs. Within a given discourse, each mention of a chain is combined with each mention of the other chains in order to yield the negative samples.

Two mentions of a pair are wrapped in `<m1> ... </m1>` or `<m2> ... </m2>` respectively. Furthermore, each line is extended by the query token `<q>` and the answer token that answers the question for relationship between the mentions: `1` for coreferent, `0` for not coreferent.

Examples for positive samples:

```
<m1> They </m1> do not work or make clothes for
<m2> themselves </m2> <eos> <q> 1

And if <m1> we </m1> do it <m2> ourselves </m2> <comma>
why should not we be able to get it done <eos> <q> 1
```

Examples for negative samples:

```
Then he began to explain everything that had been written about
<m1> himself </m1> in <m2> the Scriptures </m2> <eos> <q> 0

<m1> She </m1> saw him warming
<m2> himself </m2> by the fire <eos> <q> 0
```

The positive and negative samples are stored in two separate files.

Note that in the previous step (Section 6.2.4) we only discarded singleton-only contexts. 48% of all remaining mentions are still singletons and can only be used to generate negative samples.

6.2.6 Data Reduction

The sample sets are further reduced to subsets that meet specific criteria in order to simplify training and error analysis.

6.2.6.1 Filtering Samples by Coreference Class

We select only those samples that match a specific class of coreferences: *reflexive pronouns*, defined by the following regular expression which matches all mention openers that are followed by a reflexive:

```
<m\d+> ((my|your|him|her|it|one)self|(our|your|them|one)selves)
```

6.2.6.2 Filtering Samples by Length

Including all the resulting sequences in the data resulted in very poor performance which barely exceeded the baseline of 50% for binary classification. The reason for this is probably the vanishing gradient problem which still exists for the LSTM gates themselves (see Section 3.2).

A small-cost to huge-performance-gain tradeoff is to reduce the data to the 90th percentile in terms of sequence length. In other words, we choose a sequence length threshold that captures at least 90% of the samples (positive and negative combined). This heuristic retains 97% of the samples on average in both negative and positive sets. The maximum sequence length of the new reduced sets is 88, instead of the original 216.

6.2.7 Merging and Shuffling Samples

Note that while before filtering we had a much larger number of negative samples than positive ones, the positive subset is now slightly larger than the negative subset. This is because a large part of the chains (11%) are the only chains in their respective sentence. These sentences cannot be used to generate negative samples because there is no other chain whose mention pairs can be used to enrich the powerset of all sentence pairs (see formulae in Section 6.2.5.1).

In order to preserve the 1/1 ratio for a 50% performance baseline, we remove as many random samples from the positive subset as is necessary to synchronize its size with the negative subset. Both subsets are then merged and shuffled.

The statistics from all steps are shown in Table 6.3.

6.2.8 Masking Rare Tokens

We create a vocabulary of 3,502 top-frequent types. This is used to replace rare tokens with the special token `<unk>` (for unknown) (*masking*).

6.2.9 Creating the Data Sets

Finally, the data set is shuffled and split into 80% training samples, 10% validation samples, and 10% test samples. The final set has a vocabulary size of 3502. The training set consists of 2345 samples, the test set contains 293 samples. See Section 9.2 in the Appendix for examples from the resulting data set.

Set	elements	
contexts	25,646	
chains	60,647	
... o/w only chain in sentence	6,630	10.93%
... o/w singletons (used f.neg.samp.)	28,962	47.76%
... o/w non-singletons chains	31,474	51.90%
positive samples	51,617	
... o/w coref class subset	1,811	3.51%
... o/w in .9 length percentile	1,771	3.43%
negative samples	134,343	
... o/w coref class subset	1,529	1.14%
... o/w in .9 length percentile	1,466	1.09%
sample sentences (after merge)	2,932	
... word types	3,502	
... word tokens	102,763	
... o/w <*unk*> tokens	1,570	1.53%

Table 6.3: Statistics on sets of different stages of CoNLL-2012 preprocessing (o/w = of which).

6.3 Setup

The network performs a binary classification of each sample and optimizes its parameters by minimizing the cross-entropy between its two-dimensional output distribution and the 1-of-k target vector.

Backpropagation Through Time (BPTT) is non-truncated, i.e. goes as far as the input sequence is long. The network can handle dynamic input sequence lengths, so padding is not necessary.

All layer deltas are reset to zero after each update (full sample).

The baseline accuracy of a random binary classifier on a 1/1 mix of positive and negative samples is 50%.

6.3.1 Architectures

Deeper networks are known to better capture indirect and complex synergies between single inputs, i.e. perform higher-order abstractions. Since our task is expected to require the acquisition of latent semantic properties from raw input symbols, deepness is one of our experimental variables.

Layout	#Parameters	Layout
1L1D	18,219,282	384
1L2D	17,924,370	192*2
2L1D	16,295,698	256+128
2L2D	16,164,626	128*2+128

Table 6.4: Number of parameters for different network layouts. The hidden layer sizes are carefully chosen such that the total number of hidden nodes stays about the same.

Bidirectional networks have been shown to perform well on modeling natural language sequences (Sundermeyer et al., 2014). For our task, the question of which features to retain or learn while traversing the sequence depends on the mention or mentions to come, i.e. future features. Therefore, our training likely benefits from a bidirectional setup.

Figure 6.2 shows schematic visualizations of the network topologies, along with the labels to which the following sections refer.

6.3.2 Parameters

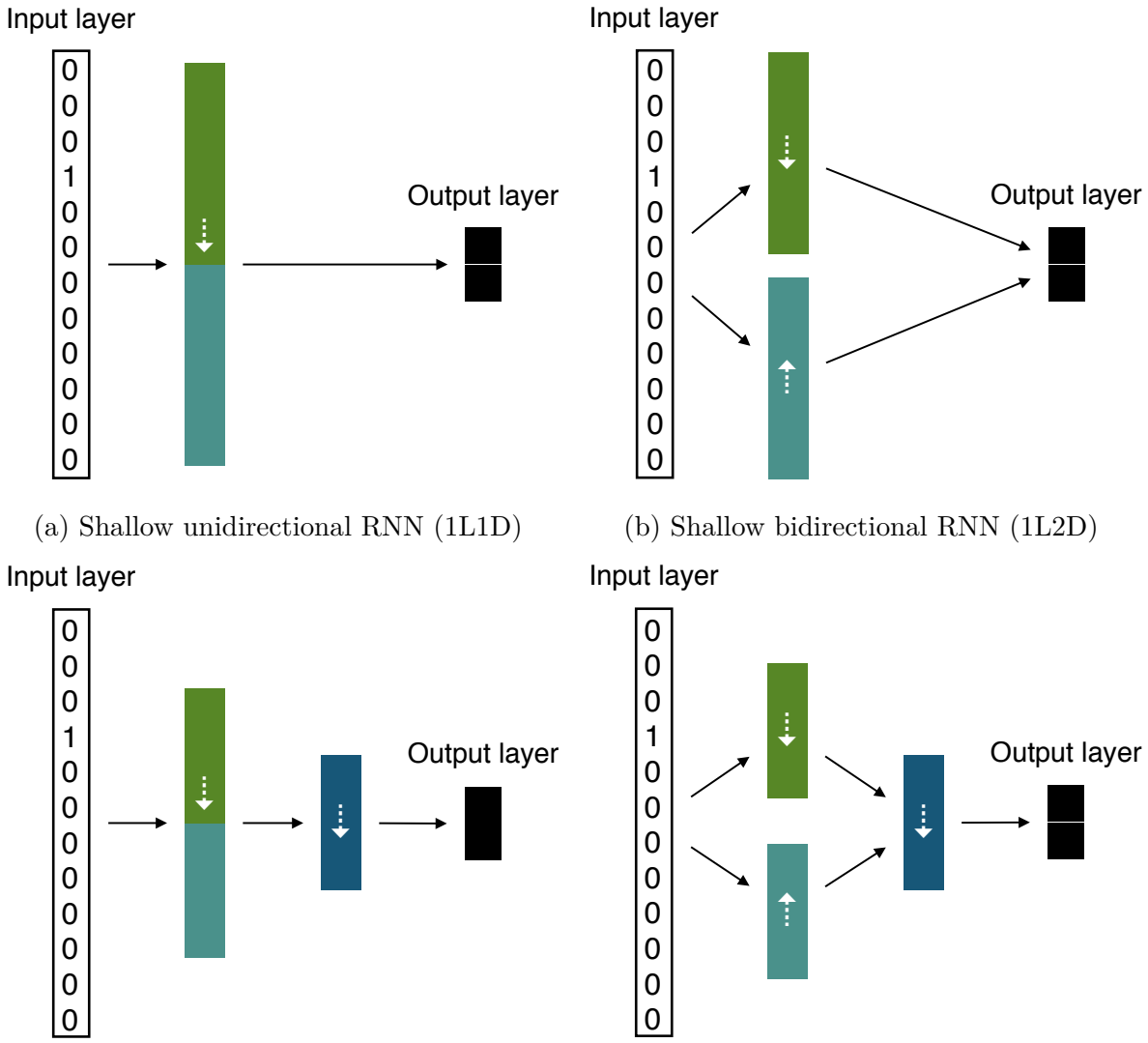
We want to make sure that the experiment runs are based on networks of similar complexities with respect to the total number of hidden nodes and parameters. In case of LSTM networks, having the same number of hidden nodes is of utmost importance, since LSTM networks use the node states to store actual bits of information.

We therefore choose to fix the total number of nodes to $3 * 128 = 384$, distributed evenly across the hidden layers. Note that we count the backward part of the bidirectional layer as an independent set of nodes. If it doesn't exist (i.e. if the layer is unidirectional), we credit its nodes to the respective container layer.

The resulting parameter distributions are shown in Table 6.4. Note that the total parameter counts shrink by ca. 2 mio. when the last hidden layer (the one connected to the vocabulary-sized output layer) becomes smaller. We accept this rather huge difference in favor of a static node count, for the reason stated above.

6.3.3 Optimization

The following experiments use RMSprop with a cache decay γ of 0.99, combined with a global learning rate α of 0.001 that is decayed by 0.97 per epoch, starting at the 10th epoch. RMSprop is known to work well without momentum (see Section 3.3.3).



(a) Shallow unidirectional RNN (1L1D)

(b) Shallow bidirectional RNN (1L2D)

(c) Deep unidirectional RNN (2L1D)

(d) Deep bidirectional RNN (2L2D)

Figure 6.2: Schematic topologies of the four architectures for the recurrent binary sequence classifier. The input layer denotes a 1-of-k encoded input to the recurrent network. The output layer is shown as a 2-dimensional vector. White downward arrows indicate forward components, white upward arrows indicate backward components of bidirectional units.

Our first experiments used a learning rate of 0.01 which turned out to be rather aggressive, especially for the large number of parameters at hand, and resulted in very unstable learning through the end. This was the case even when we decayed the learning rate by 0.97 per epoch, starting at the 10th epoch, resulting in a final learning rate of 6.5e−04 at the 100th epoch. For this reason, we used a more conservative learning rate of 0.001, both decaying and non-decaying.

As in Section 5, only the last output error is used to optimize the model, and we do not apply regularization or dropout.

6.4 Evaluation

	lr = 0.001, decay = 1			lr = 0.001, decay = 0.97		
Layout	Training	Test	p-value	Training	Test	p-value
1L1D	0.96	0.69	<0.001	0.98	0.65	<0.001
1L2D	1.00	0.74	<0.001	0.98	0.66	<0.001
2L1D	0.99	**0.75**	<0.001	1.00	0.72	<0.001
2L2D	1.00	**0.75**	<0.001	1.00	0.72	<0.001

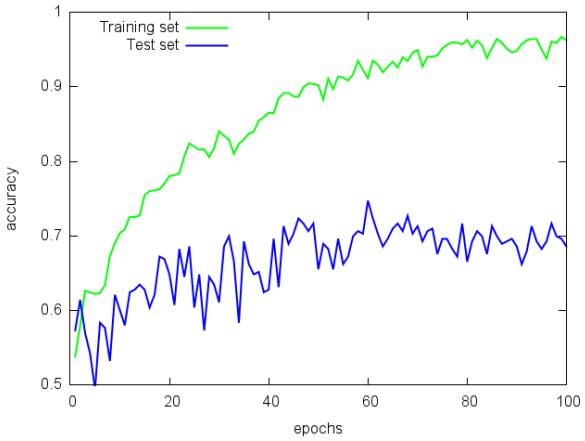
Table 6.5: Accuracies on the subset of CoNLL-2012 reduced to reflexive mentions. The p-values indicate the statistical significances of the results with respect to a random 50% prediction, as given by the binominal test. Best test set performances (in bold) are highly significant, as achieved with deep bidirectional setups and a slow conservative learning rate (i.e. no decay).

The final accuracies are shown in Table 6.5. Each result is presented with the p-value from the binominal test to determine how significant the accuracy is with respect to the baseline of 50% (null hypothesis).

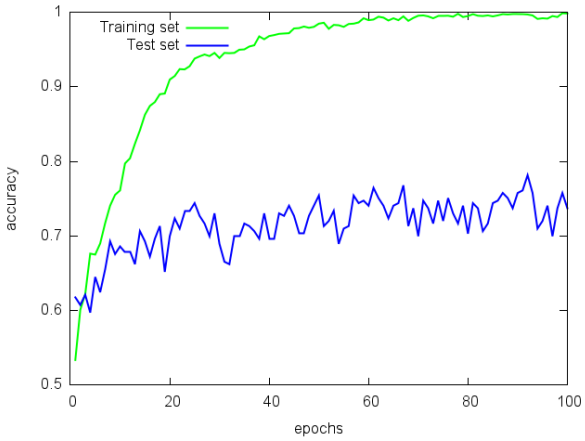
This section focuses on evaluating the performance over time because our aggressive reduction of the data set gives rise to the assumption that both learning the training set and generalizing to the test set turns out to be highly unstable. This would be indicated by high degrees of oscillation.

Training **without decay** (Figure 6.3) yields a stable and high convergence for the first time, at least for the training set. All training performances increase relatively smoothly, with a clear benefit of bidirectional layouts (Figure 6.3b, Figure 6.3d).

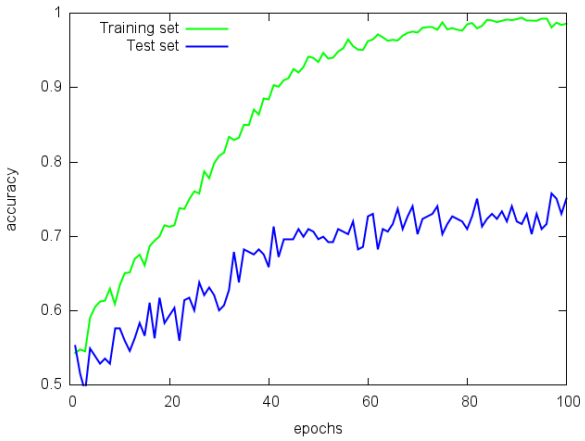
Test set performances remain rocky, but exhibit a clear decrease of oscillation in the deep bidirectional layout (Figure 6.3d).



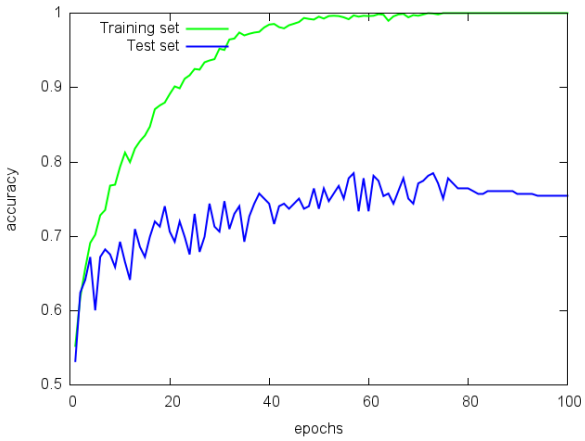
(a) Shallow unidirectional RNN (1L1D)



(b) Shallow bidirectional RNN (1L2D)

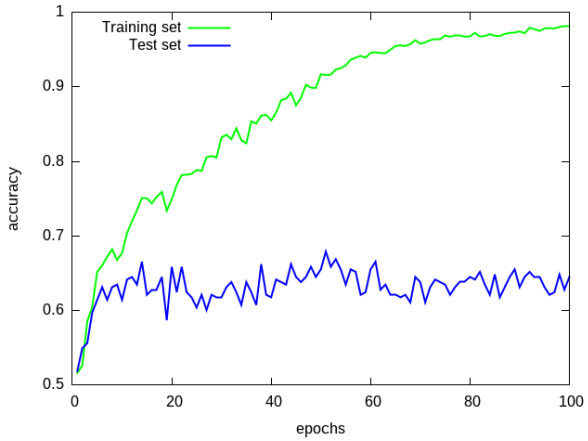


(c) Deep unidirectional RNN (2L1D)

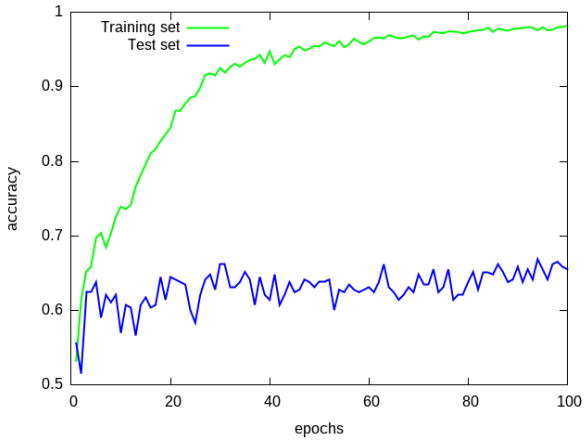


(d) Deep bidirectional RNN (2L2D)

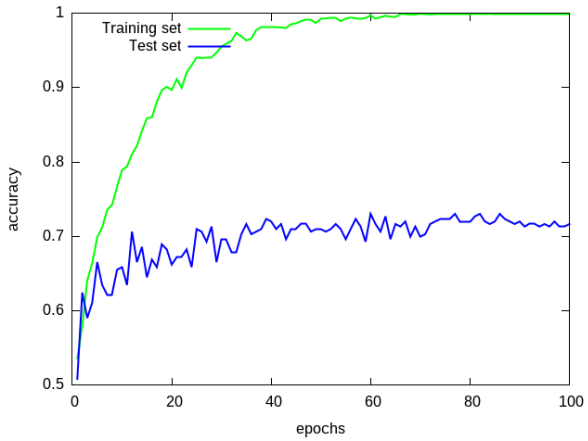
Figure 6.3: Accuracies for learning rate = 0.001 and decay = 1 (no decay)



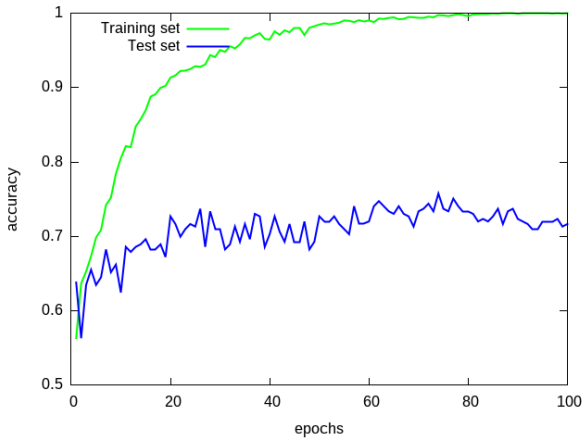
(a) Shallow unidirectional RNN (1L1D)



(b) Shallow bidirectional RNN (1L2D)



(c) Deep unidirectional RNN (2L1D)



(d) Deep bidirectional RNN (2L2D)

Figure 6.4: Accuracies for learning rate = 0.001 and decay = 0.97

Training **with decay** (Figure 6.4) produces a very similar picture for the training set performances. The only notable difference is that, compared to the previous setup, the test set performance does not converge as high. We do observe somewhat less oscillation, which hints to the fact that the network is simply losing steam with such an aggressive decay over an already low learning rate.

In summary, a deep bidirectional layout with a conservative learning and no decay yields the most powerful model.

Test accuracies are best with bidirectional layers, but do not differ much between single-layer and deep dual-layer setups. The general high or near-perfect performance of the training set shows that the network learned quite a lot from the little data and sparse representations. This gives rise to hope that the issue of overfitting can be tackled with a more appropriate setup, probably a smaller hidden size.

As expected, the performances on the aggressively reduced dataset suffer from a high degree of oscillation. This instability is somewhat solved by deeper and bidirectional models, but never quite vanishes for the test set.

One property we miss from graphs related to the decaying learning rate is that a vanishing learning rate should correlate with decreasing oscillations. No model shows this behavior, possibly because of the lack of training data from which to generalize.

Another interesting fact is that we would expect the exact same behavior across same layouts with and without decay through the 10th epoch, since we start decaying only after that. It is probable that different weight initializations are responsible for this difference in behavior. Weight initialization has been proven to be a deciding factor especially on early performances (Sutskever et al., 2013).

6.5 Linguistic Analysis

Accuracies on the high level task are a rather coarse evaluation metric, especially since the symbolic nature of the linguistic input gives us the opportunity to gain insights on a much lower level.

The following section performs a low-level analysis of the system's strengths and weaknesses regarding specific linguistic phenomena. We select the best performing model so far: dual-layer bidirectional LSTM (2L2D) with 128 hidden nodes in each layer ($128 * 2 + 128$)

Each training run produces evaluation results on the test set after the last epoch. From this evaluation, we extract the following evaluation subsets:

- **TP**: true positives (output is coreferent, target is coreferent)

- **FN**: false negatives (output is not coreferent, target is coreferent)
- **TN**: true negatives (output is not coreferent, target is not coreferent)
- **FP**: false positives (output is coreferent, target is not coreferent)

6.5.1 Definitions

Recall that each sample contains only one annotated pair of mentions, one of which is the reflexive pronoun. We thus classify each sample by a linguistic property of the other non-reflexive mention.

This approach is similar to the one in (Durrett and Klein, 2013), where the samples are additionally split along the mention status (e.g. singleton vs anaphoric) which does not apply in our data set. Stoyanov et al. (2009) performed an error analysis on more fine-grained linguistic properties.

For the sake of automation, we stick to three coarse linguistic properties which classify each mention via a top-down series of heuristics, applied to each mention string.

For these heuristics, we first define a set of Perl-standard regular expressions (case-insensitive) that describe members of the classes as follows:

Reflexives

```
$reflexives = qr/((my|your|him|her|it)self|(our|your|them)selves)/i;
```

Pronouns

```
$pronouns = qr/(I|you|he|she|it|we|they|me|him|it|us|you|them|someone
   |something|somebody|anybody|anyone|anything)/i;
```

Note the we are excluding `her` because it could be confused for a possessive pronoun.

Possessives

```
$possessives = qr/(my|your|his|her|its|our|their|someone 's|somebody␣
   's|something 's|anybody␣'s|anyone 's|anything␣'s)/i;
```

Quantifiers

```
$quantifiers = qr/(the|a|this|that|these|those|one|all|any|both|each|
   enough|every|few|fewer|little|less|lots|lot|many|more|no|several|
   some)/i;
```

Based on these definitions, the mention classes are assigned in a top-down fashion (after filtering out all reflexive mentions), with `@tokens` being the array of mention tokens. All other mentions are discarded.

6.5.1.1 Definition of Name Class

All mentions that do not consist of a quantifier or a possessive, and do not start with a pronoun, and consist only of upper case tokens, reflexives, and the genitive marker.

```
not (
    /^$quantifiers$/i
    or /^$pronouns\b/i
    or /^$possessives$/i
)
    and scalar @tokens == scalar grep { /^([[:upper:]]|'s|$reflexives
       )/␣}␣@tokens
```

6.5.1.2 Definition of Pronoun Class

All mentions that consist of a quantifier or a possessive or start with a pronoun that is not followed by a conjunction.

```
/^$pronouns\b(.})/i
    and $1 !~ /^\s(?:and|or)/
or /^$possessives$/i
or /^$quantifiers$/i
```

6.5.1.3 Definition of Phrase Class

All other mentions that consist of at least 2 tokens.

6.5.2 Examples for Mention Classes

The following sections present examples for each mention class across the four evaluation subsets.

6.5.2.1 Pronoun class

Table 6.6 presents examples of mentions for the *pronoun* class across the four evaluation subsets.

The sets feature a high degree of overlap in terms of their mentions. This could indicate that these pronominal tokens have not been memorized, but rather been used to

contribute to higher-level heuristics for classification. These heuristics in turn could have been confused by the context, possibly containing interfering pronouns.

6.5.2.2 Name Class

Table 6.7 presents examples of mentions for the *name* class across the four evaluation subsets.

Similar to the *pronoun* class, there is some overlap between the sets (`Taiwan`, `God`, `Saddam`), hinting to the context being the culprit for inaccurate classifications.

6.5.2.3 Phrase Class

Table 6.8 presents examples of mentions for the *phrases* class across the four evaluation subsets.

Both TP and TN consist mostly of mentions prefixed by an article or quantifier (e.g. `the`, `every`, `those`).

Among the true predictions, there seems to be a high degree of non-rare gendered tokens (e.g. `daughter`, `guy`, `woman`) which possibly contribute to the correct prediction. This could indicate that the model has learned some notion of gender.

6.5.3 Evaluation of Class Performances

Table 6.9 presents the mean cross-entropy errors and error variances across the four evaluation subsets.

We observe a low mean error and relatively low error variance on the TP and TN set. This indicates a high confidence of the model regarding the TP predictions. Since the sizes of TP and TN are almost equal, their differences in error and variance can be interpreted as a slightly lower confidence for the TN predictions.

As expected, false predictions correlate with high mean errors and a high variances. As observed in the previous sections, many of these false predictions stem from rare words and complex phrase mentions. More training data, and especially rich word embeddings, are obvious first solutions.

Table 6.10 presents the performances of the three mention classes for each evaluation subset. Significant deviations from a random 25% baseline distribution across the four evaluation subsets ($p < 0.05$) are marked in bold (measured via binominal test).

Subset	Mention
TP	I
	he
	his
	it
	its
	my
	our
	their
	them
	they
	we
	you
	your
TN	I
	he
	her
	his
	it
	its
	me
	my
	she
	them
	this
	us
	you
FP	I
	him
	his
	it
	their
FN	I
	her
	his
	it
	my
	she
	their
	them
	they
	this
	we

Table 6.6: Examples of mentions for the **pronoun** class across the four evaluation subsets. Note the high degree of overlap between the subsets, hinting to confusing context that prevents proper classification.

Subset	Mention
TP	Baghdad itself
	David himself
	Saddam Hussein himself
	Wang
TN	Chairman Mao
	China
	Christ 's
	God 's
	Joe
	Lone Star
	Martha Stewart
	Medicaid
	Taiwan
	Taiwan 's
	The Lord God
	The U.S.S.R.
FP	God
	Hong Kong
	John
	Li
	Mr. Barre
	Saddam
	Taiwan
FN	Eastman Kodak Co.
	Mr. Fazio
	Taiwan
	The State Department

Table 6.7: Examples of mentions for the **names** class across the four evaluation subsets.

Subset	Mention
TP	A number of his friends who had insider information
	An employer
	Apelles <comma> who has proved himself to be a true follower of Christ
	every proud idea that raises itself against the knowledge of God
	the Spirit himself
	the international community
	the office
	these guys
	this body that dies
	those who live
TN	A guy in the engineering team
	a cheap loft
	documents relating to the national security of the United States
	her brother 's
	his wife
	our sick daughter
	that cage
	the CTI
	the Chinese government
	the Victory over Japan
	the entire nation and all of the people
	the past <num> years
	the two men
	the woman who is touching him
	these shootings
	this <unk> economic growth
	this group of people
FP	the renowned author Li Ao
	the show
	the things you prepared for yourself
	them and Saddam Hussein himself
	these six cachets
	your teaching
FN	First Meridian 's president <comma> Roger V. Sala <comma>
	This group
	a small number of company bosses
	dear children
	his students
	the British government
	the Lebanese
	the US Supreme Court
	the city
	the company 's
	the planet
	the prospective nominee
	the young artists

Table 6.8: Examples of mentions for the **phrases** class across the four evaluation subsets.

Set	size	mean error	variance
TP	108	0.26	1.29
TN	104	0.99	5.89
FP	37	25.52	390.81
FN	44	18.20	129.39

Table 6.9: Mean cross entropy errors and error variances for all four evaluation subsets.

Set	size	pronoun	p-value	name	p-value	phrase	p-value
All	293	142		49		86	
TP	108	**73**	<0.001	9	<0.4	21	1
TN	104	33	<0.7	**26**	<0.001	**42**	<0.001
FP	37	**15**	<0.001	10	<0.6	**10**	<0.01
FN	44	**21**	<0.01	**4**	<0.01	**13**	<0.05

Table 6.10: Performances of the three mention classes for each evaluation subset. Significant deviations from a random 25% baseline distribution across the four evaluation subsets ($p < 0.05$) are marked in bold (measured via binominal test).

Members of the *pronoun* class are most likely to be identified as TP (51%), while still consisting of a large quantity of false predictions in FP and FN (25%).

Members of the *phrase* class consist of almost half TN samples (49%), but also noticeably of false predictions in FP and FN (27%). Interestingly enough, this behavior is similar to that of *pronoun* samples, with the roles of TP and TN switched respectively.

Members of the *name* class consist of over half TN samples (53%), combined with the smallest fraction of FN samples (8%) compared to the other classes.

In summary, the model has greater difficulties in identifying true positives for *phrase* and *name* than for *pronoun*, though even the latter still features a high degree of false classifications. Since the mentions of *pronoun* exhibit a high number of overlapping tokens (see Table 6.6), we suspect that these mentions often occur in complex and confusing context for which the lack of data prevents proper abstraction.

A possible explanation for the low performance on *phrase* and *name* is that the model fails to assign crucial features for resolving coreferences, possibly due to the large variety of names and lack of training data.

6.5.3.1 Accuracies of Mention Classes

Table 6.11 presents the accuracy for each mention class in the test set of a dual-layer bidirectional LSTM classifier. All accuracies are highly significant ($p < 0.005$) with respect

to a 50% random classification.

	pronoun	name	phrase
Accuracy	.75	.71	.73

Table 6.11: Accuracies of each mention class in the test set of a dual-layer bidirectional LSTM classifier. All accuracies are highly significant ($p < 0.005$) with respect to a 50% random classification (measured via binominal test). Mention pairs of class *name* are hardest to classify correctly. Mention pairs of class *pronoun* are easiest to classify.

It seems that mention pairs of class *name* are hardest to classify correctly, whereas mention pairs of class *pronoun* are easiest to classify.

This ranking of difficulty reflects the degree of variability we found for each class. Pronouns consist of a fixed set of tokens, whereas phrases and names hold open sets of possible values.

The fact that phrases are still slightly easier to resolve than names possibly stems from certain indicator tokens within the phrases that simplify the classification. For example, the following mentions from the true-positive set feature clear and frequent indicators of plural and neuter:

```
these guys
these people
this body that dies
```

Examples from the true-negative set feature frequently gendered tokens:

```
her brother 's
his wife
the woman who is touching him
```

Examples from the false-negative sets contain tokens whose numbers and genders should also be relatively easy to identify:

```
Mr. Fazio
Taiwan
the young artists
This group
the Lebanese
the US Supreme Court
```

The fact that the model failed to classify these mentions properly could be explained by the combination of a simple 1-of-k word representation with a severe lack of training data and thus insufficient distributional information. It is safe to assume that replacing the

1-of-k representations of the input tokens by dense word embeddings pre-trained on a large corpus could remedy many of these issues.

6.6 Conclusion

We trained several variants of a recurrent Long Short-Term Memory (LSTM) binary sequence classifier to identify true and false instances of mention pairs. Apart from an XML-style annotation of the two mentions, we did not provide any features. The recurrent framework allowed the models to take the full context into account, from beginning to end.

We reduced the CoNLL-2012 data set to a training corpus with 90% of the longest sequences, and applied a filter to pick only the samples with at least one reflexive pronoun mention. On this reduced corpus, the generic approach was able to beat the baseline on real-life data with high significance and high confidence for the true predictions, using solely sparse 1-of-k representations as input. In order to tackle more complex data, it is likely that more complex setups and rich input representations are required.

Our reduced data set allowed us to dive deeper into the evaluation of specific linguistic subsets of the test samples, each classified by the type of non-reflexive mention they feature: *pronoun*, *name*, and *phrase*. We found that our model indeed seems to learn higher-level heuristics, contrary to the previous toy data approach in Section 5. It was shown that the performance of each linguistic subset reflected the difficulty with which they are to handle. Pronoun samples were easiest to resolve, while phrases and proper names were harder to resolve.

Further experiments should tackle the following issues:

Our system currently lacks components at both ends of the Coreference Resolution (CR) pipeline. Solving the official CoNLL-2012 challenge requires initial mention detection, and the grouping of mentions to full chains. Both of these components can and probably should be tackled by external tools, so they are not part of the actual mention-pair identification system, and are thus not top priority.

More importantly, the data reduction methods lead to easy victories with generic models, but they also have a devastating effect on the already small annotated data from CoNLL-2012. The reduced corpus set has little material to generalize from, which was evident from the low validation accuracies in the performance analysis. Further experiments should aim to make the non-reduced data set work, possibly with the help of dense word embeddings trained on a larger unannotated corpus, and a more extensive search over the hyperparameters.

Furthermore, even taking the longest sentences (around 300 tokens) into account ignores the fact that coreference chains in CoNLL-2012 can span several sentences. A whole discourse typically consists of a paragraph across which coreferent mentions are linked. It is unlikely that an LSTM network of decent size is able to tackle these vast distances. One solution for this issue is a sliding window of size ρ with truncated Backpropagation Through Time (BPTT). The network can be set up to perform several updates per sample (instead of just one), where each update looks back ρ time steps within the sample. Karpathy et al. (2015) show that ρ can bet set fairly low and would still allow the Recurrent Neural Network (RNN) to capture the interdependencies between mentions that are farther away from each other that ρ.

The training speed on these larger samples can be boosted by removing the peepholes, allowing a more efficient computation of the LSTM states.

Eventually, unrestrained data will provide many more different linguistic classes of mention pairs than the three presented in this chapter. The linguistic analysis will have to be expanded accordingly. For example, one could employ a tree-based analysis in which each node corresponds to a component of the context, e.g. from discourse to sentence to different types of noun phrases, e.g. pronouns, names, and so forth. The test set could then be iteratively filtered by these criteria in order to find the exact phenomena that a model struggles with, similar to the oracle method proposed by Karpathy et al. (2015).

Chapter 7

Visual Analysis of Recurrent Neural Networks

In Section 6, we cast the problem of mention-pair identification to a binary classification model: Given a sequence of tokens with two annotated mentions, one of which is a reflexive pronoun, and a final query token, predict whether the annotated mention pair is indeed coreferent or not. This employs a Question-Answering (QA) regime within a well-defined task of a small subset of coreference phenomena We hope that this model learns explicit features instead of distributed context embeddings which are hard to interpret.

This chapter deviates from the linguistics-driven analyses of high-level performances and dives down into the low-level activations of the neural states at each step of selective epochs during training and testing. In the course of the following investigations, we try to explain high-level observations in terms of the behavioral patterns of neurons and layers within the network.

Considering the complexity of the data, and the fact that we do not yet know what pattern to look for, we strive for intuitive visualizations that are easy to inspect and, hopefully, easy to interpret.

The *Torch* code for reproducing the results in this chapter is available at:
https://github.com/kaumanns/DeepDiver

Section 7.1 provides an overview of previous work in the field of low-level network analysis.

Section 7.2 defines the methodology and terminology by which the visualizations are performed.

Section 7.3 provides and discusses visualizations for saturations of Long Short-Term Memory (LSTM) gate activations along the anchor tokens of our mention-pair-annotated samples.

Section 7.4 introduces and discusses t-SNE visualizations of the memory cell activations along the anchor tokens of our mention-pair-annotated samples.

Section 7.5 summarizes the findings and suggests further directions for the visual analysis of Recurrent Neural Networks (RNN).

7.1 Related Work

7.1.1 Performance Analysis

Chung et al. (2014) provide evidence for the general effectiveness of gating units on polyphonic music modeling and speech signal modeling. They find that Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRU) each shine at different tasks, but admit that more research is necessary on the differences between these two architectures.

Recently, Józefowicz et al. (2015) addressed the question of whether LSTM networks are a truly optimal solution to the problem of vanishing gradients, or just an expensive ad-hoc solution for which better alternatives might exist. They compare a number of alternatives with different gating mechanisms (e.g. GRU) on long-term dependency modeling tasks: language modeling, XML modeling, simple arithmetics, and a form of music modeling. The group found that while some alternatives performed better at certain tasks than the expensive vanilla LSTM, none provided consistent improvements. They conclude that if there are better algorithms, they are not trivial to discover.

A very important finding of their work was that though LSTM networks lose to some alternative architectures, notably GRU networks, adding a bias of 1 to the LSTM forget gate closes their performance gap. This is attributed to the fact that random parameter initialization effectively primes the forget gates to forget by default. Adding a bias of 1 thus boosts initial learning, leading the group to echo a recommendation that had originally been given with the introduction of forget gates by Gers et al. (2000). The LSTM architectures in this work follow this recommendation.

Recently, Greff et al. (2015) performed an extensive study on the synergies between hyperparameters on eight variants of LSTM networks by systematically changing only single variables in their setups, e.g. omission of certain gates or peepholes, omission of activation functions, and coupling of gates. The test data comprises recognition of speech and handwriting and polyphonic music modeling. For the visualization of the results, the group chose a mix of histograms, pie charts, heatmaps, and line graphs. The plots shed light on the correlations between network variants, classification errors, negative log-likelihoods, numbers of parameters, and different hyperparameters, such as learning rates

and hidden sizes. These visualizations allowed the authors to draw useful conclusions and recommendations for the design of LSTM networks.

On a similar motivation, Breuel (2015b) focuses on extensive heatmap visualizations and inspections to explore the interactions between hyperparameters in large scale experiments on MNIST data. The results are plotted via scatter plots and several different color gradients in order to show which design choices are likely to affect the performance or not. For example, it appears that batch sizes need to be combined with much smaller learning rates from a more narrow scope which greatly diminishes (or nullifies) their effectiveness. The results also show the superiority of Rectified Linear Units (ReLU) for deep architectures compared to other activation functions.

Similar techniques are used in Breuel (2015a) in order to pinpoint the effects of different hyperparameter choices on the convergence of stochastic gradient descent optimization. It is shown that parameter initialization is indeed crucial for the final convergence, though the assumed problem of local minima appears to be non-existent for these particular experiments.

The approaches outlined so far prove that a method of intuitive (i.e. visual) inspection is crucial for designing architectures and experiments. However, they focus on high performance analysis of different setups and do not further inspect exactly how these differences come into place. It is not clear whether the differences between their setups really come down to the network wiring, or if other factors spoil the results. For example, it is possible or even likely that certain types of input are detrimental to one model and beneficial to the other. These input types may not even be restricted to the different data types of e.g. visual analysis and NLP, but may also be boiled down to specific linguistic phenomena, as is done by Karpathy et al. (2015). It appears that the confidence drawn from high-level performance investigations of these kinds rely on both the scale of their evaluations (over ten thousand models in Józefowicz et al. (2015)), and grid search or random selection of hyperparameters.

7.1.2 Low-Level Analysis of Image Classifiers

Recently, Samek et al. (2015) proposed a framework to measure the importance of small regions within an image that feeds into the decision of a deep neural network image classifier. As the group points out, the lack of transparency in deep non-linear algorithms is a growing problem in designing better systems for increasingly complex tasks. Heatmaps over the input data are argued to allow unsupervised assessment of high-level performances.

This work is continued by the related group of Montavon et al. (2015) who propose the application of deep Taylor Decomposition and heatmaps to intuitively trace the relevance

of single pixels for the classification of images in a deep neural network. Their visualization technique helps pinpointing the exact aspects of the input that cause the final decision, as is shown on the established data sets of MNIST and ILSVRC. They point out that their approach can serve as a general evaluation methodology for various network tasks. However, despite these successes on tracing back visual classification, its usefulness for NLP systems is yet to be assessed.

7.1.3 Low-Level Analysis of Language Models

A prevailing question in the context of Deep Neural Networks (DNN) for Natural Language Processing (NLP) is: do the hidden layers learn explicit features, or rather distributed context representations? This is not a trivial question. The very purpose of Feedforward Multilayer Perceptron (MLP) networks and even Recurrent Neural Networks (RNN) is to pipe the input representations, sparse or dense, through an informational bottleneck from which higher-level predictions can be generalized. Latent intermediate features are derived from the data and represented in a connectionist fashion, presumably added to many other pieces of latent information. Contrary to the symbolic approach to information processing, this approach usually obscures single features from the human eye.

Palangi et al. (2015) use heatmaps over gate activations to assign single neurons of gates and cells to specific words in a sentence. The underlying model is trained to learn sentence embeddings to maximize the relevance of a document to a query. Therefore, they assume that a sudden change in the activation of a cell signifies the detection of a keyword, and thus its connection to this word. While it is an intriguing approach to restrict the visualization to quick activation changes, it is not clear how this would account for features that are represented in a distributed fashion.

Wu and King (2016) provide an analysis of the average forget gate activations of a large 256-sized LSTM network during speech synthesis. While single-neuron information lost, the analysis shows that there are clear spikes of remembering and forgetting at phoneme boundaries, proving that it is a critical component in the network.

Karpathy et al. (2015) deliver a framework for visualization that tries to trace back high-level decisions of the network to low-level activations of gates and memory layers. Their guinea pig system is a character-level language model that continuously rolls over different kinds of text, e.g. natural language and computer programs. Random picks of memory cells revealed single neurons that seem to be responsible for special tasks, such as keeping count of indents or memorizing whether the model is currently accepting input from inside parentheses. They confirm this behavior for all tested models, but do not provide an explanation. The group analyzed the ratios between open and closed gates across time, layers, and samples, and show that gates in deeper layers seem to shift from a distributed mode of operation to a more sparse one. Furthermore, they strategically

stripped away easy errors via their "oracle" mechanism (e.g. errors that can simply be tackled by more training data) in order to find out which problems truly stem from the network design, not from a lack of information.

The paper by Karpathy et al. (2015) has been one of the corner stones of this work. However, despite their successes in diving down the network, their visualizations still lack a proper linguistic analysis, for example hypotheses about how the behavior of single gates relates to the challenges of the task. This is probably at least partially due to the fact that language models are difficult to analyze because the design of the data (unannotated real-life natural language) does not allow easy alignment of possibly related states. This chapter will try to advance on these shortcomings.

7.2 Methodology

This section introduces the terminology and definitions required in subsequent chapters.

7.2.1 Terminology

7.2.1.1 Levels and layers

Conventional network terminology uses the word "layer" for each group of network units that do not directly depend on each other and can thus be computed in parallel at each step. This terminology is not only problematic when it comes to distinguishing the two components of a bidirectional Recurrent Neural Network (RNN), but also when the individual sublayers of a Long Short-Term Memory (LSTM) block have to be addressed individually.

Recall that the components of an LSTM unit are as follows (convenience names in parentheses):

- input gates (*input gate*)
- forget gates (*forget gate*)
- output gates (*output gate*)
- memory cells (*memory cell*)
- block inputs (*block input*)
- block outputs (*block output*)

Calling these components "sublayers" may make sense from a schematic viewpoint, but it is misleading with respect to the algorithm. There is no algorithmic difference between

the layers within an LSTM[1]. Each is the result of a sum of linear transformations and subsequent non-linear activation, and thus corresponds to a node in a directed acyclic graph. Note that contrary to intuition, an RNN is not a cyclic graph because its recurrent connections cannot be traversed indefinitely but must adhere to the directed graph layout of the unrolled network (see Section 3).

Furthermore, a "bidirectional layer" is implemented as two parallel LSTM units which, again, implement the same algorithm. We would have to speak of "subsublayers" in order to address e.g. the forget gate of the backward part of a bidirectional LSTM unit. It is obvious that imposing an artificial terminological tree structure on the groups of neurons of an LSTM network is an unnecessary complication and distracts from the similarity of their roles within the network. Therefore, we decide to adhere to the following terminology:

- A **neuron** is the smallest unit of a neural network, i.e. the variable that holds the activated sum of a point-wise weighted vector of size N in $\mathbb{R}^N$.
- A **layer** is an array of neurons. Each layer corresponds to a node in the computational graph.
- A **network** is composed of at least two layers (hidden and output).
- A **level** consists of one layer (as in simple RNNs) or several layers (as in bidirectional RNNs or LSTM networks). The individual *memory cell* layers of a level (e.g. hidden layers in simple RNNs and memory layers in LSTM networks) are independent from each other and can be computed in parallel (which is what effectively implements the different levels of abstraction within a neural network, hence the name). The term "level" is thus a generalization of the common notion of a "layer" which better accommodates the nested topology of non-linear units in bidirectional RNNs and LSTM networks.
- A **deep** architecture arranges at least two levels of layers.

7.2.1.2 States of a Network

A **state** denotes a concrete non-linear activation at a specific point in time. States for different components of the network are defined as such:

- A **neural state** in $\mathbb{R}$ is the state of a single neuron.
- A **layer state** in $\mathbb{R}^N$ is an array of N neural states.
- A **network state** is the concatenation of the states of some subset of layers (e.g. all *input gates* or all *memory cells*).

Each state can be addressed via a tuple of set, epoch, iteration, and time step (see Section 7.2.2).

[1] Quite contrarily, there is a closer functional similarity between layers of the same type across LSTM blocks.

7.2.1.3 Labels

We keep the plots simple in terms of diversity of labels and colors. Our goal is to show that intuitive visualizations with well-chosen labels can provide useful insights about the inner workings of an RNN. Using more than two, maximum three labels would require further systematic cluster analysis to prove that two groups of data points are indeed significantly separate.

We will use the following terminology for the three different layer types in our model:

- *1-FWD* for the first-level forward part of the bidirectional LSTM
- *1-BWD* for the first-level backward part of the bidirectional LSTM
- *2-UNI* for the second-level unidirectional LSTM

Figure 7.1 outlines the network topology of the deep bidirectional LSTM network which is used for the analysis in this chapter. The training log of that model provides activation states for each layer of each LSTM block, across all iterations and time steps.

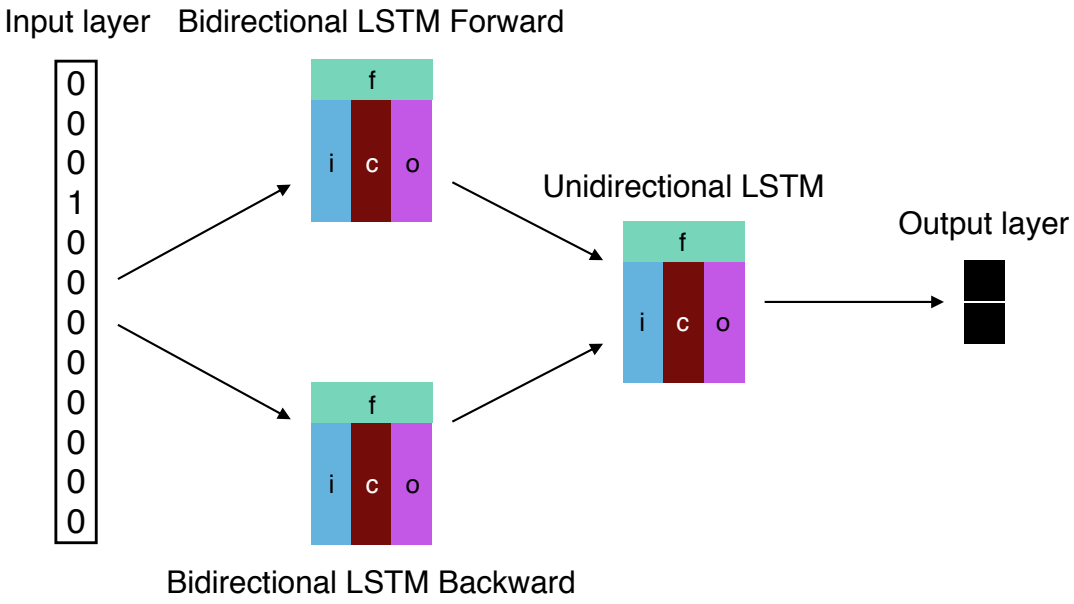


Figure 7.1: Topology of deep bidirectional LSTM binary classifier whose states are analyzed in this chapter. Each rectangle represents an LSTM unit, comprising four layers for which states are saved in the log: *input gate* (i), *forget gate* (f), *output gate* (o), and *memory cell* (c).

7.2.2 Framework

The following section describes the framework by which we generate our plots.

7.2.2.1 Step 1: Collect Network States

Definitions

Each network state is specified by a 4-tuple of $\langle set, epoch, iteration, step \rangle$:

- $set \in \{test, valid\}$ (unshuffled)
- $epoch \in \{1, ..., 100\}$
- $iteration \in \{1, ..., length(set)\}$, where $length(set)$ is a function of the set that returns its number of samples.
- $step \in \{1, ..., length(\langle set, epoch, iteration \rangle)\}$, where $\langle set, epoch, iteration \rangle$ specifies a sample and $length(\langle set, epoch, iteration \rangle)$ is a function of that sample that returns its sequence length.

State Filters

We use the Kleene star ($*$) for a position in a tuple to signify all possible values. The term *anchor tokens* denotes the five special markup tokens that are shared between all samples of the data: `<m1>`, `</m1>`, `<m2>`, `</m2>`, and `<q>`.

We apply several filters to the set specified by the 4-tuple of network states, defined as follows:

- For *set*:
  - $set \in \{test\}$
- For *epoch*:
  - $epoch \in \{100\}$
    (i.e. last epoch, fully trained model)
- For *iteration*:
  1. $iteration \in \{x | x \in \{\langle set, epoch, * \rangle\}\}$
     (i.e. all 293 iterations of that epoch)
  2. $iteration \in \{x | x \in \{\langle set, epoch, * \rangle\},$
     x.output = x.target$\}$
     (i.e. all 212 (72%) true positives/negatives))
- For *step*:
  1. $step \in \{x | x \in \{\langle set, epoch, iteration, * \rangle\}\}$
     (i.e. all steps)

2. $step \in \{x | x \in \{\langle set, epoch, iteration, * \rangle\},$
   $x.token \in \{<m1>, </m1>, <m2>, </m2>, <q>\}\}$
   (i.e. all steps at anchor tokens)

7.2.2.2 Step 2: Concatenate Layer States

Definitions

The LSTM network spans two levels comprising a total of 3 LSTM blocks (2 bidirectional blocks in the first level, 1 unidirectional block in the second level), each with 4 layers.

Each layer state within a network state is specified by a 3-tuple of $\langle level, type, layername \rangle$:

- $level \in \{1, 2\}$
- $type \in \{fwd, bwd, uni\}$
- $layername \in \{ingate, forgetgate, outgate, cell\}$

For each network state there is a total of $3 * 4 = 12$ layer states.

Layer Subsets

As in Section 7.2.2.1, we choose different subsets of the layers of the LSTM network:

- $level$: all
- $type$: all
- $layername$:
  1. $layername \in \{ingate, forgetgate, outgate, cell\}$
     (for saturation plots in Section 7.3)
  2. $layername \in \{cell\}$
     (for state scatter plots in Section 7.4)

For each network state, we simply concatenate the states of its *layername*s, resulting in a matrix of real number data points where:

- each row represents a network state
- each column represents a neuron in one of the concatenated *layername*s

7.2.2.3 Step 3: Categorize Data Points

After the previous steps, each resulting data point is labeled by a category by which it is colored in the final plot. Using the proper categorization for a data set is crucial in order to give a solid base for the emerging visual evidence.

Each network state is categorized by a combination of the input token and the values from one the following categories:

- binary **output** in $\{output0, output1\}$
- binary **target** in $\{target0, target1\}$
- **nesting** in $\{nested, separate\}$

The values of the binary output/target category denote a positive or negative sequence classification respectively. The values of the nesting category denote whether the two mentions of the sample are nested or separate (see Section 7.4.4.3 for details).

7.3 Gate Saturations

The following section picks up on the work of Karpathy et al. (2015), reproducing and extending their analysis of Long Short-Term Memory (LSTM) gate saturations. Consequently, we first outline the differences between our two architectures.

In the remainder of this chapter, we will refer to the character-level language model of Karpathy et al. (2015) as CharRNN, and to the many-to-one sequence classifier implemented in Section 6 as SCRNN.

7.3.1 Differences Between CharRNN and SCRNN

A CharRNN continuously rolls over single characters of a text. Single characters lack rich meanings. Higher-level features, if necessary, have to be derived from a sequence of characters whose meaningful boundaries are not yet known. This allows a great deal of flexibility, but also makes learning and analysis more difficult. As with all language models, the task primarily encourages the accumulation and continuous application of distributed features of the context.

Since it is hard to group single steps in the sequence into larger categories, e.g. by linguistic features, such a network can only be probed across all available time steps. Our SCRNN, in contrast, receives independent sequences of symbols with variable lengths.

The problem is framed as a many-to-one classification task, performed by a bidirectional LSTM unit in the first level and a unidirectional LSTM unit in the second level.

The LSTM character-level language model by Karpathy et al. (2015) (CharRNN) stacks a third LSTM component on top. However, it does not yield higher gate saturations than the second one, which is why we decided to rather invest into a bidirectional component in the first layer of our sequence classifier.

Features: The model is expected to learn useful features from the sequence pattern in order to solve the task. It is possible that these features correspond to historically well-defined linguistic categories, though this would be learned via patterns of correlation alone. As discussed in Section 4.1.2, even simple statistical correlations provide valuable information for resolving coreferential relations.

Anchor Tokens: Each sample contains the same five anchor tokens at variable positions within the sequence. The four mention markers indicate beginning and end for each of the two coreferential entity candidates respectively. The query token finalizes the context and triggers the prediction. Since anchor tokens are positioned in a consistent order across the samples, they allow us to probe the network at time steps that correspond to each other.

7.3.2 Definition of Gate Saturations

Recall that the states of single gate neurons are defined in Section 5.6.2 as follows:

- A *sigmoid*-activated gate neuron is open when its state is > 0.9.
- A *sigmoid*-activated gate neuron is closed when its state is < 0.1.

We introduced two **policies** as meaningful descriptive terms for the role of a single gate at a single time step:

- ***conservative*** gates inhibit the flow of information from upstream to downstream (*input gate*, *output gate*) and protect or retain the current *memory cell* states (*forget gate*).
- ***progressive*** gates allow the current *memory cells* to change in the light of new information (*input gate*, *forget gate*) and let information flow further downstream (*output gate*).

Still, it is difficult to talk about the general behavior of gate neurons across many time steps, or the general policy of a whole layer of gates. For this reason, we further abstract the notion of policies for single gates at single time steps to **saturations** of gates and layers across time, inspired by the definitions by Karpathy et al. (2015):

- A ***saturated*** gate or gate layer is mostly either open or closed.
- An ***unsaturated*** gate or gate layer is mostly neither fully open nor fully closed.

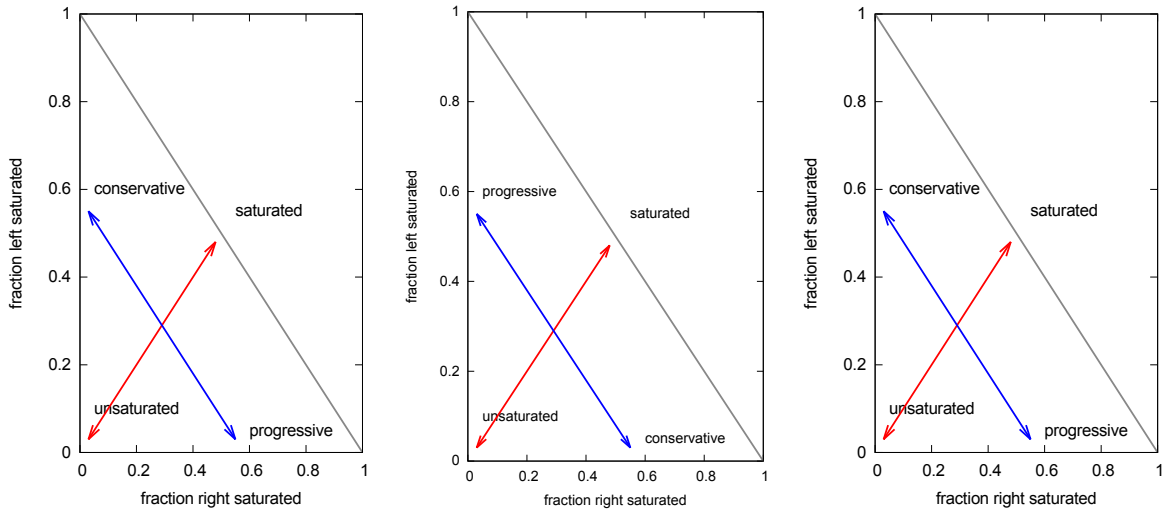
The policy and saturation for each neuron are plotted along the axes of a two-dimensional *saturation plot*. Each combination of these two variables corresponds to a rough area in the graph.

Figure 7.2 shows schematic orientations for policy and saturation. Gate neurons that are more *saturated* tend toward the 1-diagonal marker. Gate neurons that are more

unsaturated tend toward the origin of the graph. Neurons of *input gate* (Figure 7.2a) and *output gate* (Figure 7.2c) that are *progressive* tend toward the right, and toward the left if they are *conservative*. Neurons of *forget gate* (Figure 7.2b) that are *progressive* tend toward the left, and toward the right if they are *progressive*.

Examples:

- An *input gate* neuron that is *unsaturated* and slightly *conservative* is positioned somewhere near the middle of the left axis.
- An *output gate* neuron that is *saturated* and neither *progressive* nor *conservative* is positioned close to the 1-diagonal near the center of the graph.



(a) *input gates* (b) *forget gates* (c) *output gates*

Figure 7.2: Schemes for the policies and saturations of each gate. The labels denote the general regions to which a gate of that policy would tend.

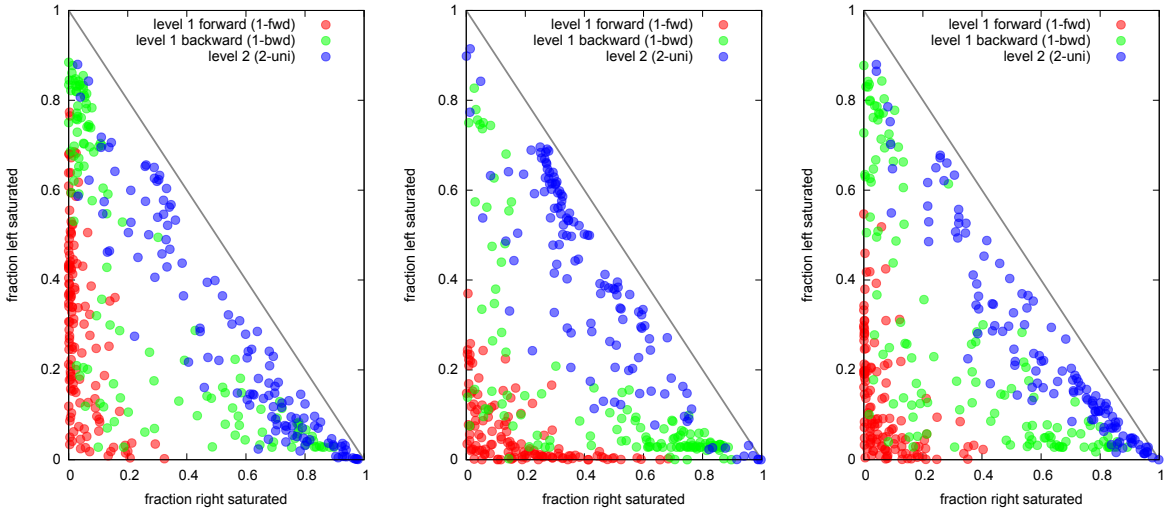
7.3.3 Hypothesis

We define our initial hypothesis as such: Assuming that our network learns sparsely represented features, contrary to distributed ones, each gate is fully saturated, i.e. assumes only open or closed states. This would place all points along the grey diagonal where the fractions sum up to one. The following saturations plots show a clear deviation from this initial hypothesis.

We will evaluate explanations and provide more fine-grained analysis. Each graph contains three times 128 plot points, one point for each gate neuron.

7.3.4 Saturations at All Time Steps

We first replicate the original saturation plots from the character-level language model RNN (CharRNN) in (Karpathy et al., 2015), derived from our SCRNN. We therefore probe all gate states across all iterations and time steps in the test set.



(a) *input gates* (b) *forget gates* (c) *output gates*

Figure 7.3: Gate saturations at all tokens across unfiltered test set (10500 network states with 128 neurons per layer). Each point represents a gate neuron. The position on the x-axis indicates the fraction of times it spends right-saturated (> 0.9). The position on the y-axis indicates the fraction of times it spends left-saturated (< 0.1). The grey diagonal indicates where the fractions would add up to one.

Figure 7.3 shows the saturations of each neuron for all tokens (i.e. all time steps) across all 293 test iterations, encoded as TP/TN/FP/FN-filtered set, or simply *unfiltered set*. The unfiltered set provides 10500 data points for each neuron.

We can confirm that the second level (*2-UNI*) tends to be more saturated in general. However, where Karpathy et al. (2015) found that the first level is struggling with a clear saturation for any node, we find clear tendencies for *1-FWD*, and even more extreme saturations for *1-BWD*.

Furthermore, our *1-FWD* neurons do not aggregate in a thick *unsaturated* lump at the center. *1-FWD* neurons are very *unsaturated* and exhibit a particular strategy of avoidance. They cautiously tend to be *progressive* (Figure 7.3a and Figure 7.3b), or show no tendency at all in the *output gate* (Figure 7.3c).

1-BWD neurons are more extreme with strong *saturated* conservative tendencies (Figure 7.3a and Figure 7.3b). This could hint to a minor importance of the backward part

of the bidirectional LSTM unit.

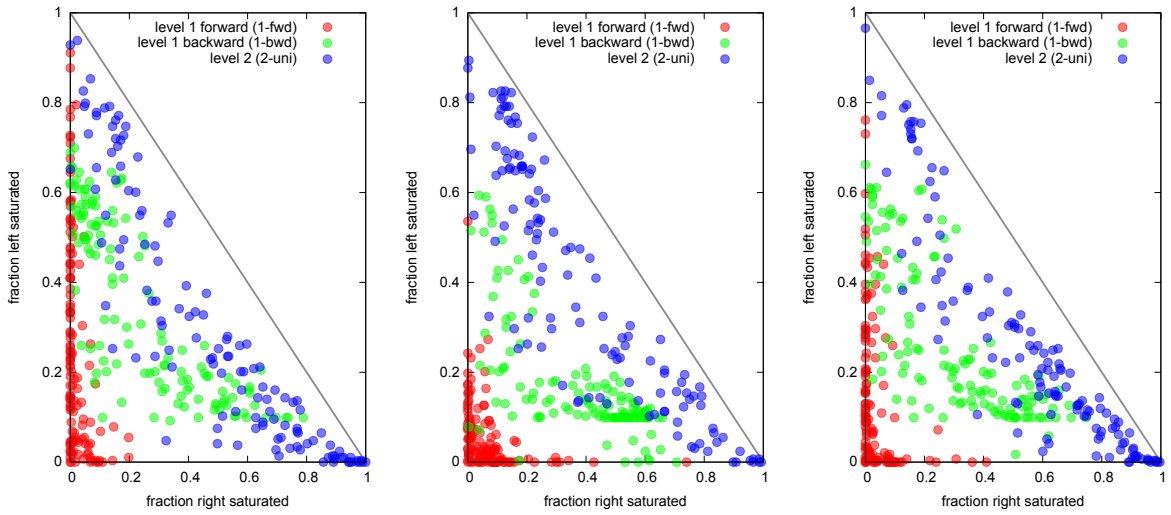
2-UNI neurons are similar to their equivalents in Karpathy et al. (2015), but more extreme. Each forget gate neuron is wildly retaining and deleting *memory cell* contents (Figure 7.3b), employing a *saturated* policy with a slight *progressive* tendency. Input Gates (Figure 7.3a) and *output gates* (Figure 7.3c) are mostly *progressive* which, in our many-to-one training regime, only enables information transfer between time steps.

Overall, both first-level layers are similarly conservative, and the second level is fairly progressive.

Coming from this general overview, the following sections probe the network inside the sequences along the anchor tokens. We will evaluate the hypothesis that a SCRNN indeed learns latent distinct features from the sequence and carries them over to the query token.

7.3.5 Saturations at Mention Markers

The following section will discuss the evolution of saturations across the mention markers of the samples.



(a) *input gates* (b) *forget gates* (c) *output gates*

Figure 7.4: Gate saturations at *<m1>* across unfiltered test set (293 network states with 128 neurons per layer). Each point represents a gate neuron. The position on the x-axis indicates the fraction of times it spends right-saturated (> 0.9). The position on the y-axis indicates the fraction of times it spends left-saturated (< 0.1). The grey diagonal indicates where the fractions would add up to one.

After **the opening first mention** (Figure 7.4), we see a clear scale of saturation from *1-FWD*, *1-BWD*, to *2-UNI* where it approaches the 1-diagonal. Else, the layers are neither *conservative* nor *progressive*. Note that the bottom-left lump of *1-FWD* reminds of the one in Karpathy et al. (2015).

Curiously enough, there seem to be consistent aggregations of *1-BWD* for the lower right corner of the *forget gate* and the *output gate*. This means that two groups of *1-BWD input gates* and *output gates* are consistently closed about 10% of the times. In other words, the backward *forget gates* seem to erase a specific subset of the *memory cell* layer, while the backward *output gates* seem to inhibit a specific subset of *memory cells*.

These consistent aggregations could be a hint to two groups of *memory cells* that represent an important latent feature from the future sequence. The closed *forget gates* possibly tell the model to act on the upcoming first mention in a specific way, such as discarding assumptions about which upcoming features to retain. The closed *output gates* in turn, influenced by the same future information, inhibit certain features from being carried back further in time.

If this is true, then it is possible to retrieve the exact respective 10% of the 293 samples that cause these gates to close down. Further investigations could easily find the common properties between the retrieved samples and pinpoint the feature represented by the respective group of *memory cells*.

After **the closing first mention** (Figure 7.5), the changes are surprisingly humble.

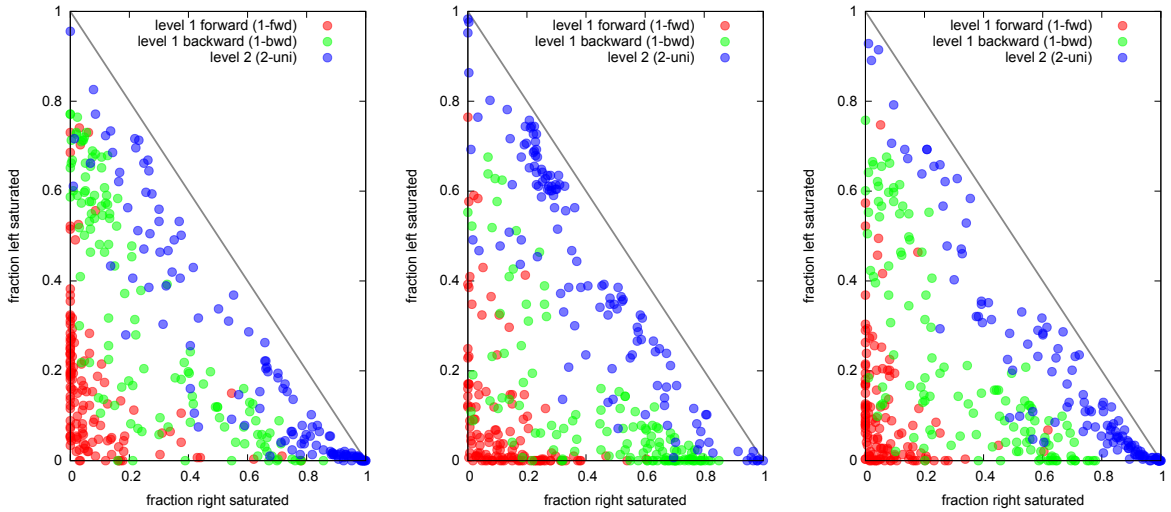
The *forget gates* of *1-BWD* have become a bit more conservative, while the other gates of *1-BWD* are now more *progressive* (Figure 7.5a, Figure 7.5c), shifting lumps of neurons toward the right-saturated edge (Figure 7.5b).

1-FWD has become a bit more *unsaturated* about its slight *progressive*ness across all gates, as can be seen by the thicker lumps at the graph origins.

After **the opening second mention** (Figure 7.6), *1-FWD* has become less *conservative* and a bit more *saturated*, spending more time in saturated states. *1-BWD* is still a mixed bag with slightly more *progressive* behavior. *2-UNI* has started aggregating in the strongly *progressive* corner for *input gates* and *output gates*, while being more mixed with respect to retaining and forgetting *memory cell* contents. It will stay in this pattern through `</m2>`.

After **the closing second mention** (Figure 7.7), *1-BWD* has become strongly *conservative* in the *input gate*, obviously not expecting relevant input. Else it adheres to the role of a *conservative* bidirectional backward component and keeps remembering future cell states.

1-FWD has suddenly shifted the policy of its outgate from non-*progressive* to non-



(a) *input gates* (b) *forget gates* (c) *output gates*

Figure 7.5: Gate saturations at *</m1>* across unfiltered test set (293 network states with 128 neurons per layer). Each point represents a gate neuron. The position on the x-axis indicates the fraction of times it spends right-saturated (> 0.9). The position on the y-axis indicates the fraction of times it spends left-saturated (< 0.1). The grey diagonal indicates where the fractions would add up to one.

conservative. With respect to the first-level layers, this time step is surprisingly unrevealing, except for their overall proximity to the patterns we observe at <q>.

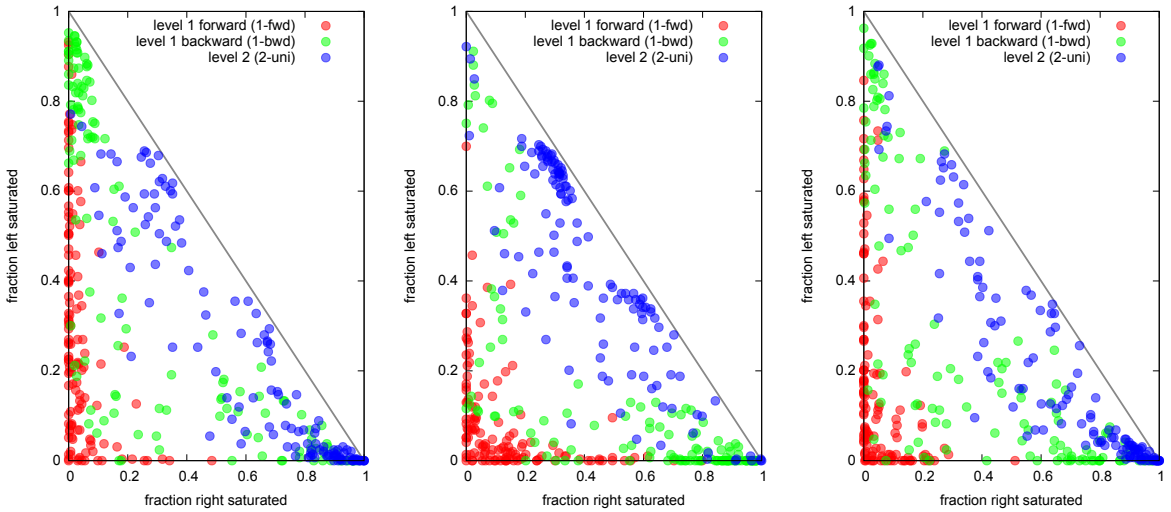
2-UNI has gradually built up to this step and stayed true to its policies from earlier steps. It is the only one of the three layers that will radically switch at the query token.

7.3.6 Saturations at Query Tokens

Figure 7.8 shows the same saturation plots as before, but restricted to final states of each iteration, i.e. at query token <q>. Since our network is trained as a many-to-one sequence classifier, we expect a clear difference between the saturation patterns at the query token and the mention markers.

1-FWD exhibits fairly similarly scattered saturations across the gate types, with tendencies to either axis. As in Section 7.3.4, a neuron is either not *conservative* or not *progressive*, and never very *saturated*.

Also as before, *1-BWD* is fairly *conservative* in ingate and forgetgate (Figure 7.8a and Figure 7.8b), though not as extreme as across all time steps. The interesting difference is



(a) *input gates* (b) *forget gates* (c) *output gates*

Figure 7.6: Gate saturations at <*m2*> across unfiltered test set (293 network states with 128 neurons per layer). Each point represents a gate neuron. The position on the x-axis indicates the fraction of times it spends right-saturated (> 0.9). The position on the y-axis indicates the fraction of times it spends left-saturated (< 0.1). The grey diagonal indicates where the fractions would add up to one.

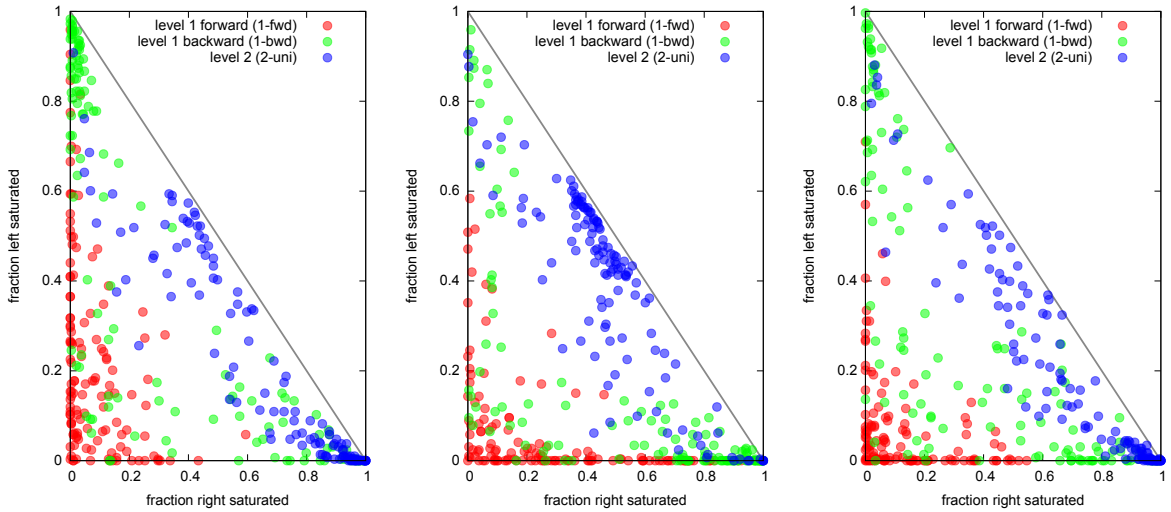
in the *output gate* where it features a fairly thick lump in the closed section, and a clear array of mostly open neurons (Figure 7.8c). It has become more *saturated*.

The trophy for the most radical change of mind goes to *2-UNI*. Not only does it more aggressively close down gates, which leads to both locking out input and forgetting *memory cell* states. It also shows clear aggregations toward the center, especially for the *forget gate* and *output gate* (Figure 7.8b and Figure 7.8c).

In other words, *2-UNI* shifted to a much more saturated regime. In general, the gates seem to have adopted a *conservative* policy, i.e. shutting out information (*input gate*) or retaining it (*forget gate*). It is plausible that what makes *2-UNI* gates so *saturated* is that its gates are selectively operating on single *memory cells* that are part of a specific feature, or even represent a full feature by themselves.

7.3.7 Summary

We confirmed that gates in the forward part of the bidirectional component tend toward lower saturations. As in CharRNN, the saturations of our model increase with deeper layers.



(a) *input gates* (b) *forget gates* (c) *output gates*

Figure 7.7: Gate saturations at </m2> across unfiltered test set (293 network states with 128 neurons per layer). Each point represents a gate neuron. The position on the x-axis indicates the fraction of times it spends right-saturated (> 0.9). The position on the y-axis indicates the fraction of times it spends left-saturated (< 0.1). The grey diagonal indicates where the fractions would add up to one.

However, there are some notable differences in the results of analysis of our model.

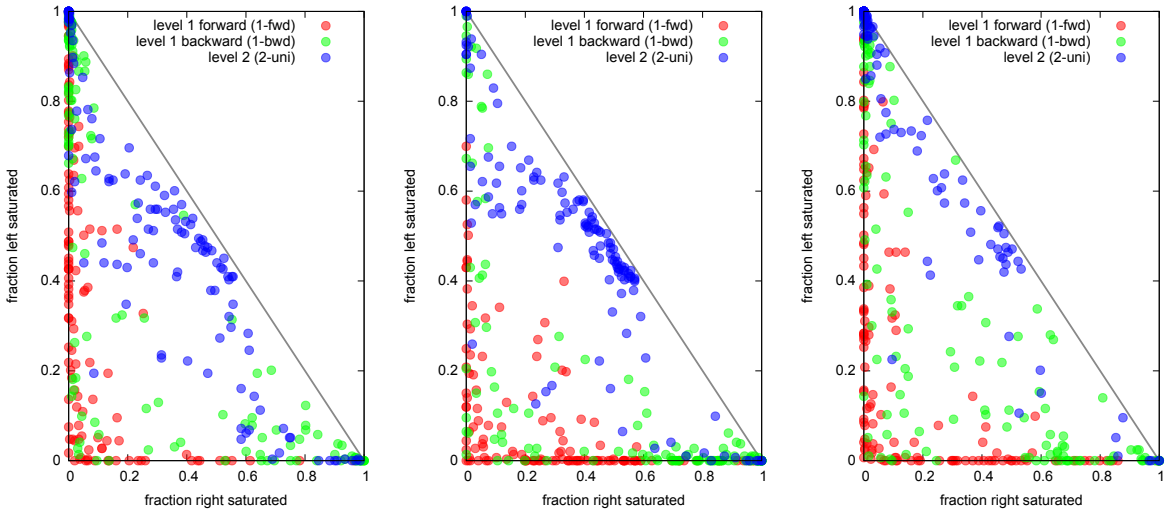
7.3.7.1 More extreme states in deeper layers

In general, our model features more *saturated* gates with higher saturations than in Karpathy et al. (2015). We also found a greater tendency for whole layers to be either very *conservative*, very *progressive*, or both.

This general tendency toward extremes is likely due to the nature of our binary classification task for CR. This task can be solved by learning and utilizing specific features, rather than computing correlation-based synergies between many latent features in a language model.

The behavior observed could be explained by such sparsely represented features that cause the model to selectively switch *memory cells* on and off in order to solve the task. If and how saturated gates control specific features is subject to further investigations.

Further experiments should reveal whether these higher saturations are also fueled by our rather large memory layers of 128 neurons each. As the same information would have to be squeezed into fewer cells, it is reasonable to assume that smaller layers would push the



(a) **Ingate** at $<q>$ (b) **Forgetgate** at $<q>$ (c) **Outgate** at $<q>$

Figure 7.8: Gate saturations at $<q>$ across unfiltered test set (293 network states with 128 neurons per layer). Each point represents a gate neuron. The position on the x-axis indicates the fraction of times it spends right-saturated (> 0.9). The position on the y-axis indicates the fraction of times it spends left-saturated (< 0.1). The grey diagonal indicates where the fractions would add up to one.

degree of saturation more toward that of a language model.

7.3.7.2 Higher saturations in backward component

Our saturations do not only increase with deeper layers, but also in the backward part of the bidirectional component, compared to the forward part. It seems that higher saturation is not an exclusive feature of deeper layers, i.e. of higher-level representations.

It remains to be investigated whether this means that higher saturation can be triggered by two different dynamics here, or it is caused by a common phenomenon.

7.3.7.3 Increasing saturation with time

The bulk of gate neurons acquire an increasingly saturated state in the course of the sequence. Again, this applies especially for the first-level backward part and the second level.

In contrast, the first-level forward part seems to work off a more basic representation level. It employs a less saturated regime with a slight tendency to spread across the

conservative edges of the graphs.

Most surprisingly, and contrary to Karpathy et al. (2015), we did find a few *memory cells* that are consistently revealed or blocked at certain steps of the sequence, hinting to specific features being learned. These extremes are probably hard to find in a continuous language model, such as CharRNN, because it does not provide anchor tokens at which the network can be probed.

7.4 States of Memory Cells

This section investigates common patterns between network states at different time steps.

7.4.1 Dimensionality Reduction via t-SNE

We render each network state as a two-dimensional *t-distributed stochastic neighbor embedding* (t-SNE) (Van der Maaten and Hinton, 2008). t-SNE minimizes the sum of Kullback-Leibler divergences over a set of data points, here network states, via stochastic gradient descent. The resulting points can be interpreted as the relative distances between the network states.

Contrary to Principal Component Analysis (PCA), t-SNE is a nonlinear dimensionality reduction technique and thus better able to capture complex synergies between the dimensions.

Since the loss function of t-SNE is non-convex, different hyperparameters and different initializations can lead to vastly different results. In order to keep the plots comparable, we use one single learning rate and a fixed seed for random initialization for the *sklearn.manifold.TSNE* package from the *scikit-learn* machine learning library for Python (Pedregosa et al., 2012). The training is initialized with a so-called "random state" of 0 and a learning rate of 100. In addition, we use a PCA initialization to ensure global stability and reproducibility of the t-SNE models. All other hyperparameters are left at their default values.

The resulting models are visualized in two-dimensional scatter plots, each labeled (and colored) according to different criteria. We label only by two categories (e.g. the two values of the network output) to ensure visual clarity.

7.4.2 Filtered vs Unfiltered Data

We retrieve the test set evaluation at the last epoch (100), containing 293 iterations, each with its own sequence of time steps.

For some analysis, this set is further reduced to those iterations that yielded either a true positive or true negative result. We call this the *filtered* subset, contrary to the *unfiltered* subset that contains all samples.

7.4.3 Cell States at All Time Steps

As a first approach, we plotted the t-SNEs for *memory cell* states at all time steps (i.e. all tokens) (Figure 7.9).

Figure 7.9a depicts network states from the *unfiltered* subset, derived from a total of 10500 data points. Figure 7.9b depicts only the *filtered* test iterations, resulting in a total of 7509 data points. Both plots are labeled by the outputs of the samples.

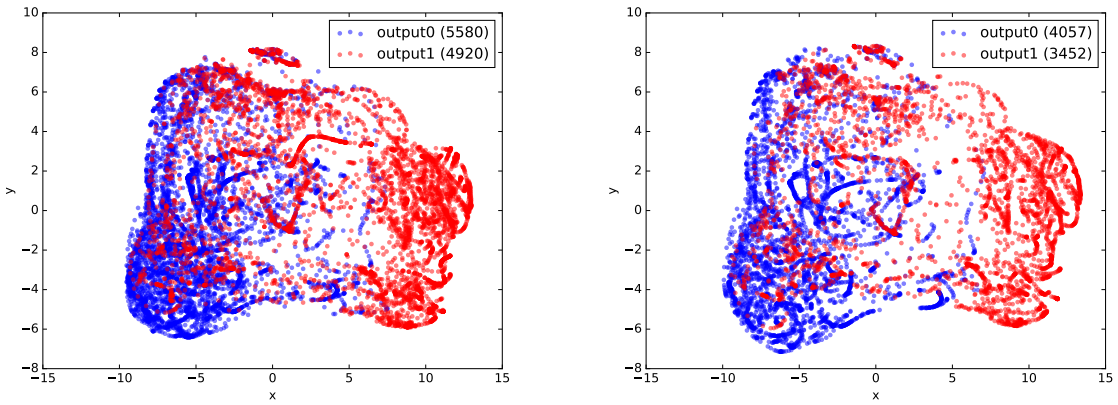
Apart from a general tendency to form fuzzy clusters of mostly 0 and mostly 1 outputs, these plots are not too revealing. This was probably the reason why Karpathy et al. (2015) had troubles selecting insightful states from their CharRNN. On average, the specific behavior of the *memory cells* at each step becomes obfuscated.

Furthermore, since both CharRNN and our model make use of variable sequence lengths without padding, it is hard to align *memory cells* of corresponding time steps, if they exist. Fortunately, our mention-pair data features five distinct anchor tokens which serve as alignments points: the four mention markers and the query token. We therefore have the opportunity to probe our model at specific steps within the sequence.

7.4.4 Cell States at Anchor Tokens

Figure 7.10 and Figure 7.11 show the unfiltered *memory cell* states at the anchor tokens, colored by output or target. They are discussed in Section 7.4.4.1. These states are the ones that lead up to the query token state plotted in Figure 7.10e and Figure 7.11e.

Figure 7.12 shows plots for the *filtered* iterations. Since the plots for outputs and targets have same shapes and colors, we only show the plots for the outputs. They are discussed in Section 7.4.4.2. These states are the ones that lead up to the query token state plotted in Figure 7.12e.



(a) Cell states at all tokensunfiltered

(b) Cell states at all tokens*filtered*

Figure 7.9: States of *memory cell* layers at **all tokens** from the test set, labeled by **output**. Each data point represents a sample, represented by three concatenated *memory cell* layer states and reduced to two dimensions via t-SNE. Each data point represents a layer state, reduced to two dimensions via t-SNE.

7.4.4.1 Labeled By Target and Output for Unfiltered Samples

Figure 7.10 and Figure 7.11 depict the *memory cell* states after the anchor tokens in the unfiltered test set, respectively labeled by output and target. Naturally, for two scatter plots at the same time step labeled by target and output, the data point positions are the same, while their differing colors show the discrepancy between true and false predictions.

After the **opening first mention** (Figure 7.10a, Figure 7.11a), both output and target states look fairly similar and unevenly scattered, indicating that the network is not quite confident about which prediction to derive from them. This is no surprise, since not enough information has yet been presented.

After the **closing first mention** (Figure 7.10b, Figure 7.11b), there is already a clear favorite side for either color, though they are still mixed. It is possible that these states are retrieved from iterations with a clear and unambiguous (w.r.t. coreference resolution) linguistic class of first-mentions, maybe pronouns.

After the **opening second mention** (Figure 7.10c, Figure 7.11c), two distinct clusters are starting to appear. Despite the separation, the target-colored cluster remains tainted by states that will ultimately trigger a false prediction (Figure 7.11c).

After the **closing second mention** (Figure 7.10d, Figure 7.11d), the separation is nearly complete, with most of the states that relate to the same output placed in the same cluster (Figure 7.10d). However, there are still some rogue states remaining in the wrong cluster.

The fact that these states will eventually be resolved until the query token indicates that the network is still expecting relevant input outside the mention tokens.

After the **query token**, all states are separated into their respective output clusters (Figure 7.10e). Samples that trigger a false prediction can easily be discerned in Figure 7.11e as blue dots in the red cluster and vice versa.

7.4.4.2 Labeled By Output for Filtered Samples

Figure 7.12 depicts the *memory cell* states after the anchor tokens in the *filtered* test set. As output and target would yield the same colors, we show only plots for the former.

After the **opening first mention** (Figure 7.12a), we observe largely the same pattern of wide dispersion as in Figure 7.10a.

After the **closing first mention** (Figure 7.12b), the two clusters are a hint more separated than in Figure 7.10b.

After the **opening second mention** (Figure 7.12c), the output labels are clearly more separated than in Figure 7.10c. Apparently, a large part of the decision has already been settled at this step, which is possibly enabled by the bidirectional layout which makes the network aware of what is about to come.

After the **closing second mention** (Figure 7.12d), we observe two distinct clusters for the two outputs, as expected, though the degree of separation is surprisingly low compared to the previous step.

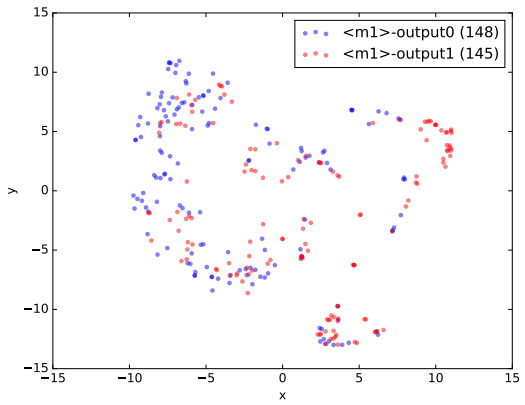
Most notably, the clusters are not yet (almost) linearly separable as in Figure 7.12e, although the tokens after the last mention marker are supposedly useless of the pair identification. Apparently, the remaining time steps serve to finalize the distinction of the clusters in order to leverage a most confident prediction, supporting a similar observation we made in Section 7.4.4.1.

After the **query token**, (Figure 7.12e) the clusters are distinctly separated, as expected from a convergent model.

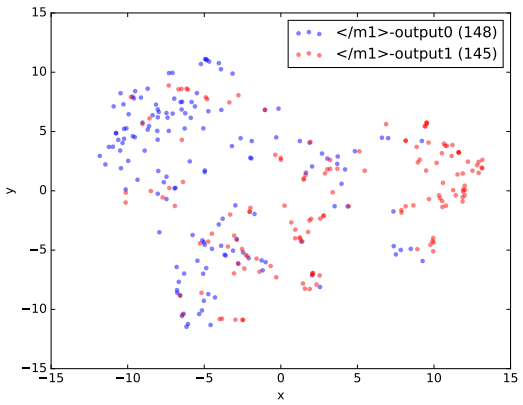
7.4.4.3 Labeled By Nested Mentions for Filtered Samples

Some of the test samples feature a nested mention chain, i.e. `<m2>` inside an `<m1>`. Example:

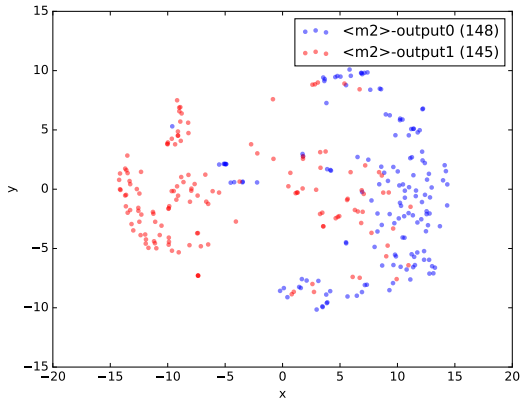
```
<m1> he <m2> himself </m2> </m1>
```



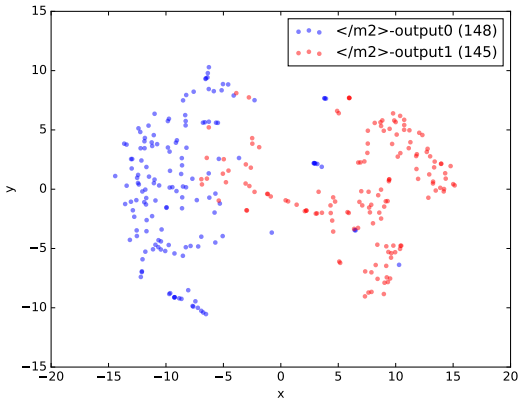
(a) Cell states at *<m1>*



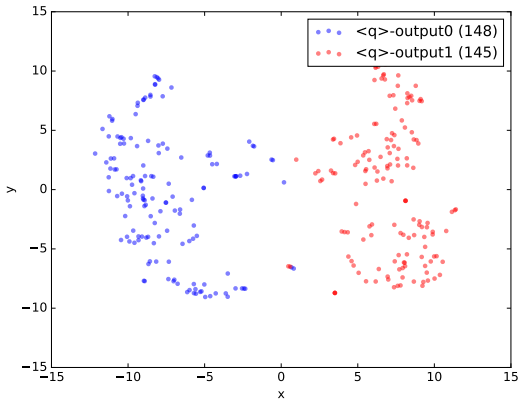
(b) Cell states at *</m1>*



(c) Cell states at *<m2>*

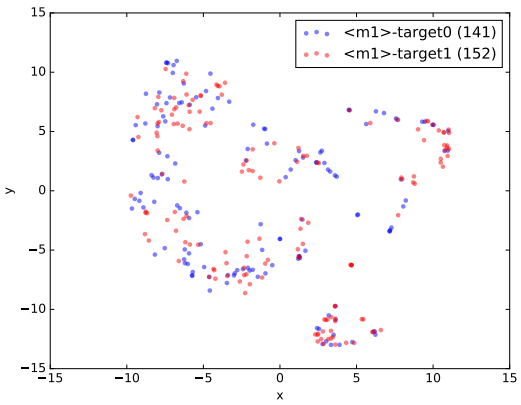


(d) Cell states at *</m2>*

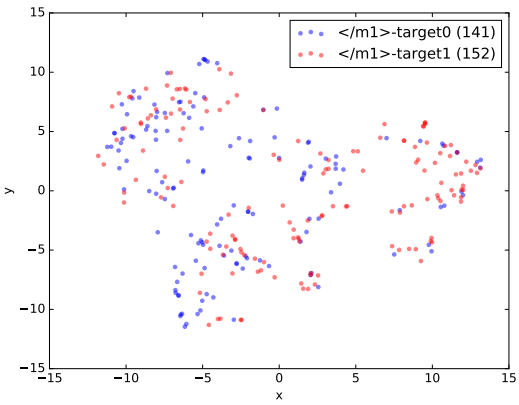


(e) Cell states at *<q>*

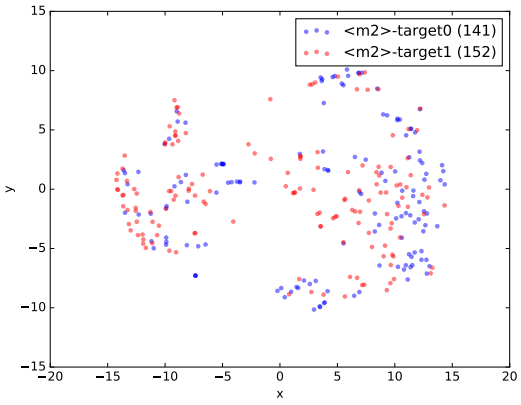
Figure 7.10: States of *memory cell* layers at **anchor tokens** from the **unfiltered** test set, labeled by **output**. Each data point represents a sample, represented by three concatenated *memory cell* layer states and reduced to two dimensions via t-SNE.



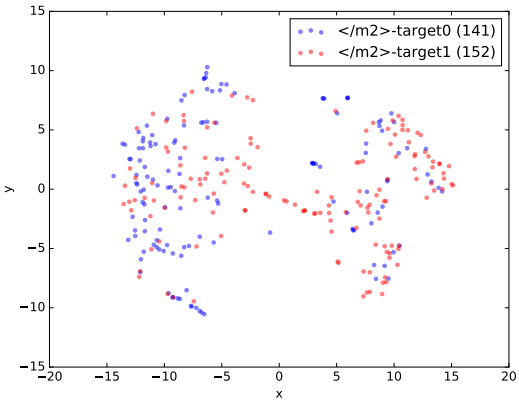
(a) Cell states at *<m1>*



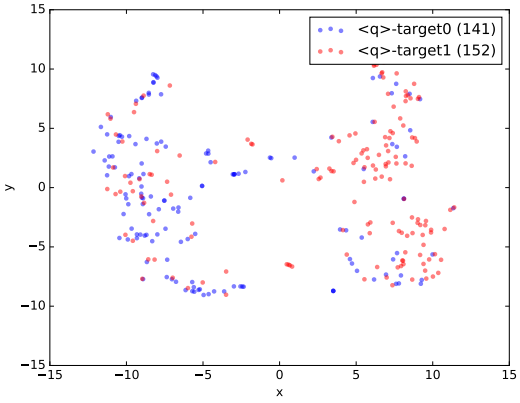
(b) Cell states at *</m1>*



(c) Cell states at *<m2>*

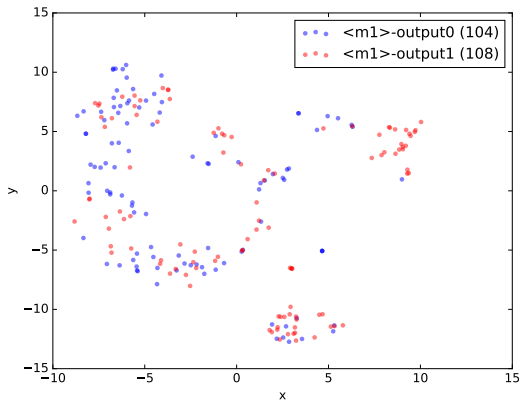


(d) Cell states at *</m2>*

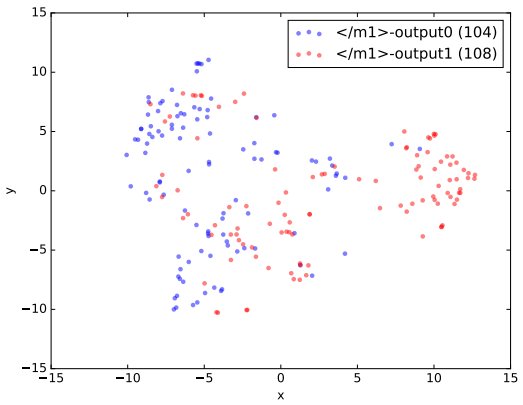


(e) Cell states at *<q>*

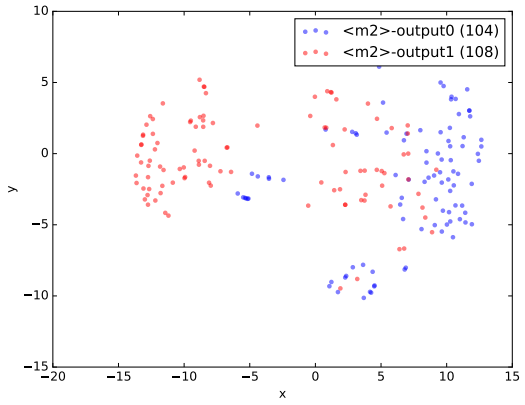
Figure 7.11: States of *memory cell* layers at **anchor tokens** from the **unfiltered** test set, labeled by **target**. Each data point represents a sample, represented by three concatenated *memory cell* layer states and reduced to two dimensions via t-SNE.



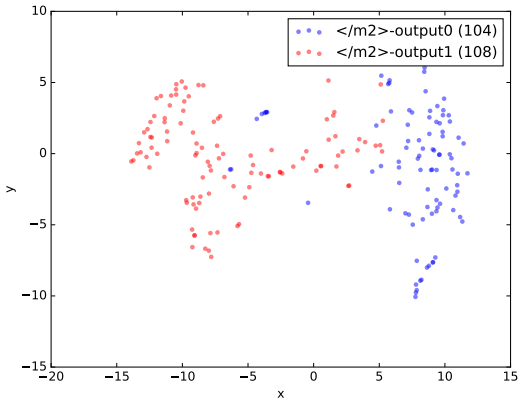
(a) Cell states at *<m1>*



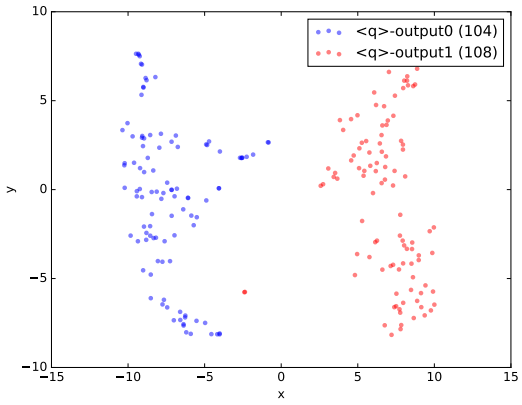
(b) Cell states at *</m1>*



(c) Cell states at *<m2>*



(d) Cell states at *</m2>*



(e) Cell states at *<q>*

Figure 7.12: States of *memory cell* layers at **anchor tokens** from the ***filtered*** test set, labeled by **output**. Each data point represents a sample, represented by three concatenated *memory cell* layer states and reduced to two dimensions via t-SNE.

Figure 7.13 depicts the *memory cell* states after the anchor tokens in the *filtered* test set, labeled by *nested* and *separate* respectively.

We expected all states related to the same *nested* feature to aggregate in the same cluster, which is indeed what we find. Moreover, the clustering of the nested states reflects the reversed order of the two closing mention markers (`</m1>` and `</m2>`).

After the **closing second mention** (Figure 7.13d), the states of nested chain samples are still fairly scattered.

After the **closing first mention** (Figure 7.13b), they aggregate clearly at the right side of the plot. Recall that in the previous section, the states at this point are still quite dispersed.

It seems that the network was able to detect nested constellations and treat them differently from the standard order of the mentions.

7.5 Conclusion

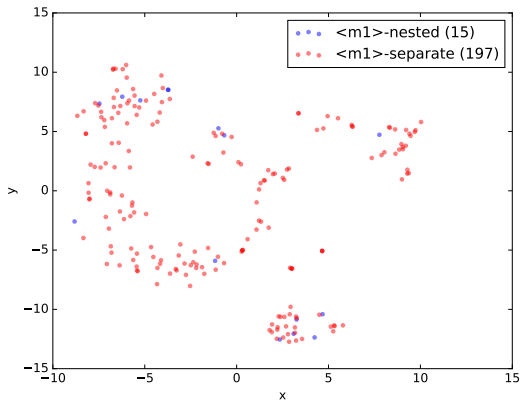
We applied two methods of visual analysis to the states produced by a recurrent LSTM binary sequence classifier for mention-pair identification, as described in Section 6.

Gate saturation plots, first described by Karpathy et al. (2015), show the average policies of single gates across the sequences, separated by gate type and colored by network level. We were able to reproduce the results and observed a general tendency for gates to become more saturated in deeper stages of the network.

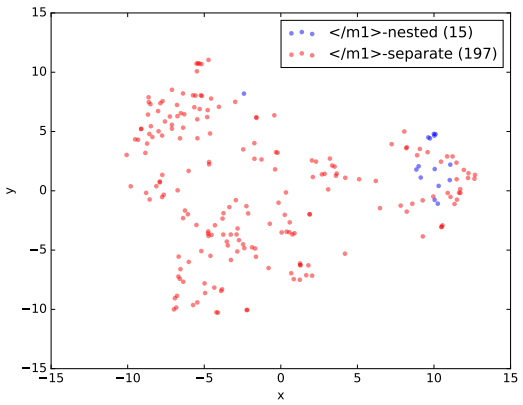
Moreover, since our data annotation comprised five anchor tokens which are present in all samples, we were able to probe the network at certain time steps corresponding to opening mentions, closing mentions, and the query token. This allowed us to confirm a consistent behavior of the gates at these time steps and to draw interpretations.

The same method of probing the model at anchor tokens could be applied to the t-SNE manifold visualization where we mapped sets of data points, each representing the concatenation of all *memory cells* of a time step, to a two-dimensional scatter plot. We showed that there is a clear progression in the pattern of *memory cell* states. The degree of dispersion of the states gradually decreases with respect to the output of the associated iteration, resulting in two distinct clusters. The fact that this shape could be reproduced across several t-SNE training runs with different steps, labels, and filters proves the robustness of this method.

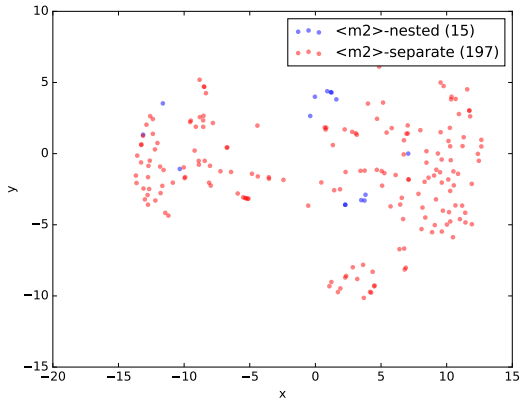
Further t-SNE *memory cell* visualizations will have to apply the same distinction between layer types as for the saturation plots. It is likely that there will be significant differences



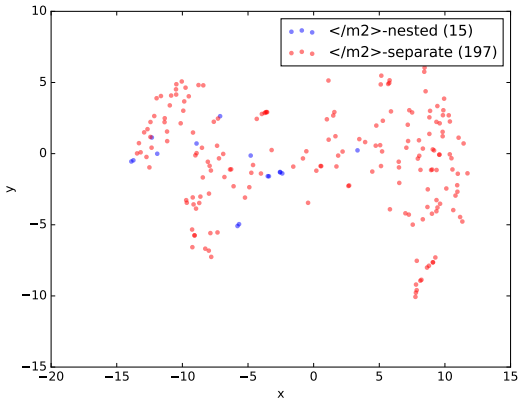
(a) Cell states at *<m1>*



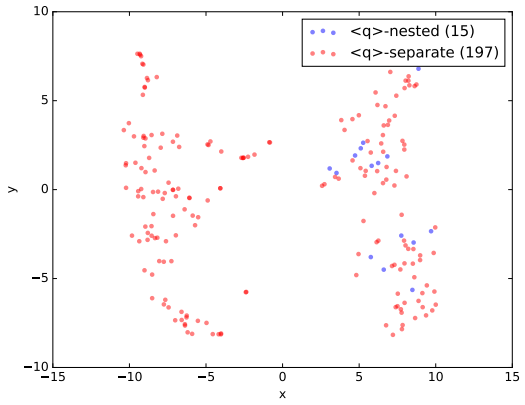
(b) Cell states at *</m1>*



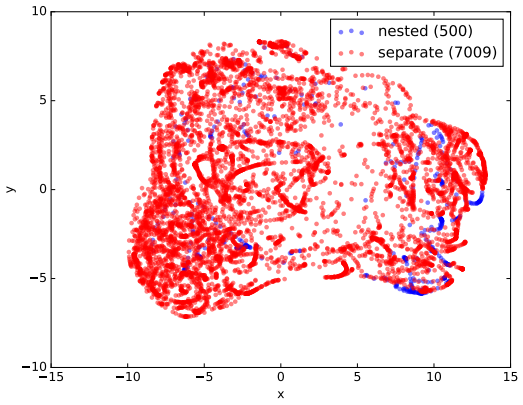
(c) Cell states at *<m2>*



(d) Cell states at *</m2>*



(e) Cell states at *<q>*



(f) Cell states at all tokens

Figure 7.13: States of *memory cell* layers at **anchor tokens** from the ***filtered*** test set, labeled by **nested**. Each data point represents a sample, represented by three concatenated *memory cell* layer states and reduced to two dimensions via t-SNE.

between the *memory cell* states of each network level.

Moreover, it could prove insightful even for larger networks to investigate heatmaps over single neurons, as in Section 5. Heatmaps have already been established as the preferred method to let the neurons paint a picture because they are able to help pinpoint attentional spikes (Hermann et al., 2015; Bahdanau et al., 2014) and shed light on the semantics of single neurons in a recurrent network (Karpathy et al., 2015). However, doing this for larger networks trained on complex tasks usually proves difficult. *Memory cells* tend to be part of a distributed representation in the memory layer. Finding a meaningful *memory cell* neuron among hundreds or thousands that employs a clear meaning is either impossible or subject to luck (Karpathy et al., 2015).

In order to make the inspection of single *memory cell* neurons feasible for larger networks, one could apply correlation analyses on the *memory cells* across the anchor tokens of all samples. This could reveal bundles of neurons that commonly "fire together", i.e. exhibit the same relative states. If connectionist models truly learn by the Hebbian rule, these bundles could provide valuable hints to latent features learned by the network. /

Chapter 8

Conclusion

8.1 Summary

This thesis explored the intricacies and possibilities of diving deep down to the neural activations in a recurrent Long Short-Term Memory (LSTM) network. We trained sequence classifiers for Anaphora Resolution (AR) and pair-wise Coreference Resolution (CR) in order to find new ways of gaining insights into what makes a connectionist model of Natural Language Processing (NLP) tick.

Section 5 discussed the implementation of a head noun predictor for AR. A synthetic corpus of simple discourses allowed us to precisely define features we expected the network to learn, and consequently look out for them in the results. We showed how a neural analysis via intuitively colored heatmaps over gates and memory cells uncovers interesting patterns and inspires future modifications to the network.

Section 6 applied a range of LSTM network architectures to the task of binary mention-pair identification. We reduced CoNLL-2012 to a subset of shorter samples employing a specific linguistic phenomenon: a reflexive pronoun for one of the two annotated mentions per sample. We showed that a generic deep bidirectional architecture performs well at this task, even when only little data and sparse word representations are given. The subsequent linguistic analysis broke the test set down into three linguistic classes and showed how the properties of each class are reflected in its performance.

Section 7 created several visualizations to the log data generated from a deep bidirectional LSTM model that was applied to the task of the previous chapter. We discussed related work in this field and the shortage of systematic low-level analyses of Deep Neural Networks (DNN) for Natural Language Processing (NLP) driven by a linguistic methodology. We exploited the fact that our training corpora feature the same five anchor tokens

for each sample, mention markers and query token, along which the data points for the different visualizations can be aligned. We probed the model at these anchor tokens and produced gate saturation plots and t-SNE manifolds of memory layers.

We confirmed that the gates of LSTM networks indeed exhibit higher saturations in deeper layers, though different layer types play an important role as well. The results support the idea that deep models are indeed beneficial for creating higher-level abstractions. This has long been know for other tasks, especially models of visual processing, but is still disputed for solving complex reasoning tasks for Natural Language Understanding (NLU).

It is evident that the gating mechanisms of LSTM units greatly help understanding the complex dynamics of an RNN, thanks to their roles as controllers over single memory cells.

8.2 Future Work

Future work on the matter and motivations of this thesis will have to focus on scaling up the complexity of the experiments and make the jump to exhaustive and real-life corpora. More specifically, we have to verify that the findings and methods developed in this work can indeed be extrapolated to large scale data sets and varied problems of NLU and Question-Answering (QA). Though working on tiny toy tasks is insightful and certainly fun, it will eventually be subjected to the same criticism of non-scalability as were models of symbolic Artificial Intelligence.

In order to achieve this goal, utilizing the full range of modern network technologies, such as Encoder-Decoder networks, attention mechanisms, and Memory Networks, is inevitable. Furthermore, pre-trained word embeddings are a mandatory part of these endeavors in order to fight the problem of scarcity of annotated data for NLU tasks.

Scaling up the complexity of the task, technology, and data, provides us with the perfect incentive to develop new visualization methods. For example, an immediate next step could be to perform correlation analyses over the neural activations and find groups of related neurons that might represent a latent feature. If the Hebbian rule of learning is indeed the underlying principle in connectionist models, these analyses are almost bound to provide interesting results.

Another important though often neglected frontier of DNNs for NLP is the expansion to new languages, especially those that employ different morphosyntactic frameworks than the English language. It seems obvious that much of the work on statistical and connectionist NLP has so far focused on the properties of the analytical and isolating properties of English. An example for this trend is the long reliance on n-gram based language models, including neural ones, who require few lexemes per word and a rigid

word order. Even improvements on this method, such as the skip-gram model, are not offering a solution to the challenges posed by synthetic and agglutinating languages.

Recent successes of character-based recurrent models promise a robust approach to these issues, but have to be investigated, and equally importantly, be analyzed down to the neuron. If the field of Artificial Neural Networks is expected to take further leaps and possibly to help reverse-engineer biological cognition, thorough analysis and investigation is an indispensable part of the scientific method.

Chapter 9

Appendix

9.1 Prolog Code for Generating Synthetic Corpus

```
:- set_prolog_flag(verbose, silent).
:- initialization main.
main :- current_prolog_flag(argv, _), run, halt.
main :- halt(1).

run :- phrase(s,L), atomic_list_concat(L,'␣',S), write(S), nl,
    fail.
run :- true.

m(M, POS) -->
    m_token(M,'<m'), call(POS), m_token(M,'</m').

m_token(M,S) --> { atom_concat(M,'>',X), atom_concat(S,X,W) }, [W
    ].

either(A,_) --> call(A).
either(_,B) --> call(B).

unique([]).
unique([H|T]) :-
    not(member(H,T)),
    unique(T).

% Grammar

s -->
```

```
25      pn(Gender1, Word1), conj, pn(Gender2, Word2), { Word1 \= Word2
            }, eos,
26      poss(pl), n(Gender3, Word3), adv, v,
27      anaphora_hub([Gender1, Word1], [Gender2, Word2], [Gender3,
           Word3]).
28
29 anaphora_hub([Gender1, Word1], [Gender2, Word2], [Gender3, Word3])
       -->
30      (
31          { Gender1 = Gender2 }, !,
32          anaphora(Word2, Gender2)
33      );
34      (
35          anaphora(Word1, Gender1)
36      );
37      (
38          anaphora(Word2, Gender2)
39      );
40      (
41          anaphora(Word3, Gender3)
42      ).
43
44 anaphora(Word, Gender) --> pron(Gender), trigger, n(Gender, Word).
45 anaphora(Word, Gender) --> pron(Gender), trigger, pn(Gender, Word)
       .
46
47 % Vocabulary
48
49 bos --> ['<s>'].
50 eos --> ['<eos>'].
51 trigger --> ['<coref>'].
52
53 pn(m, Word) --> { Word = 'Frank' }, [Word].
54 pn(m, Word) --> { Word = 'John'  }, [Word].
55 pn(m, Word) --> { Word = 'Peter' }, [Word].
56 pn(f, Word) --> { Word = 'Mary'  }, [Word].
57 pn(f, Word) --> { Word = 'Susie' }, [Word].
58
59 n(n, Word)  --> { Word = 'dog' },   [Word].
60 n(n, Word)  --> { Word = 'cat' },   [Word].
61
62 adv --> ['really'].
63 adv --> ['usually'].
64 adv --> ['very␣often'].
```

```
65 adv --> ['after␣a␣while'].
66
67 v --> ['likes'].
68 v --> ['loves'].
69
70 pron(m) --> ['him'].
71 pron(f) --> ['her'].
72 pron(n) --> ['its'].
73
74 poss(pl) --> ['their'].
75
76 conj --> ['and'].
```

9.2 Example Sentences from CoNLL-2012

Example sentences from the reduced CoNLL-2012 data set, categorized by evaluation subset and mention class.

9.2.1 True Positive Evaluation Set

9.2.1.1 Pronoun Class

```
Then <m1> you </m1> will save <m2> yourself </m2> and those who
   listen to your teaching <eos> <q>

When that happens <comma> <m1> I </m1> will draw all people to <m2>
   myself </m2> <eos> <quot> <q>

And then <m1> he </m1> must be allowed to defend <m2> himself </m2>
   against their charges <eos> <quot> <q>
```

9.2.1.2 Name Class

```
So <m1> Jesus <m2> himself </m2> </m1> became like them and had the
   same experiences they have <eos> <q>

After moving to the unoccupied zone <comma> <m1> Wang </m1> began
   carving seals in his spare time to support <m2> himself </m2> <eos
   > <q>

If <m1> Abraham </m1> was made right by the things he did <comma> he
   had a reason to boast about <m2> himself </m2> <eos> <q>
```

9.2.1.3 Phrase Class

```
And <m1> this body that dies </m1> must clothe <m2> itself </m2> with
    something that will never die <eos> <q>

He died for all so that <m1> those who live </m1> would not continue
   to live for <m2> themselves </m2> <eos> <q>

Give greetings also to my dear friend Stachys and to <m1> Apelles <
   comma> who has proved <m2> himself </m2> to be a true follower of
   Christ </m1> <eos> <q>
```

9.2.2 True Negative Evaluation Set

9.2.2.1 Pronoun Class

<m1> He </m1> says he <quot> thought Clinton could handle it <m2> himself </m2> <eos> <quot> <q>

<m1> I </m1> do not mean that we are able to do anything good <m2> ourselves </m2> <eos> <q>

Even nature <m1> itself </m1> teaches <m2> you </m2> that wearing long hair is <unk> for a man <eos> <q>

9.2.2.2 Name Class

I <m1> myself </m1> have seen <m2> Christ 's </m2> sufferings <eos> <q>

The greatest person in <m1> God 's </m1> kingdom is the one who makes <m2> himself </m2> humble like this child <eos> <q>

<m1> Martha Stewart </m1> told me that the investigation <m2> itself </m2> had cost her about <punct> <num> million <comma> in decline in her stock <comma> legal fees <comma> and lost business opportunities <eos> <q>

9.2.2.3 Phrase Class

<m1> That truth </m1> is Christ <m2> himself </m2> <eos> <q>

He secretly kept some of <m1> the money </m1> for <m2> himself </m2> <eos> <q>

Well actually Condi Rice <m1> herself </m1> in her own confirmation hearing had rattled <m2> that cage </m2> <eos> <q>

9.2.3 False Positive Evaluation Set

9.2.3.1 Pronoun Class

John <m1> himself </m1> will go ahead of the Lord and make people ready for <m2> his </m2> coming <eos> <q>

They made a promise to <m1> themselves </m1> that they would not eat or drink anything until they had killed <m2> him </m2> <eos> <q>

To do a better job and <m1> I </m1> believe that we would 've done a better job we would 've again found <m2> ourselves </m2> in the arena of <punct> <q>

9.2.3.2 Name Class

<quot> <m1> Going </m1> to work every day is a responsibility I owe <m2> myself </m2> <comma> <quot> says Yeh <eos> <q>

But the Pharisees and experts in the law refused to accept God 's plan for <m1> themselves </m1> <comma> they did not let <m2> John </m2> baptize them <eos> -RRB- <q>

If you think you can fool <m1> God </m1> <comma> you are only fooling <m2> yourselves </m2> <eos> <q>

9.2.3.3 Phrase Class

People still fear <m1> them and Saddam Hussein <m2> himself </m2> </m1> <eos> <q>

Then you will save <m1> yourself </m1> and those who listen to <m2> your teaching </m2> <eos> <q>

When I saw <m1> these six cachets </m1> <comma> my first reaction is that it is absolutely hopeless to rely on the government <m2> itself </m2> to control institutional thickening <eos> <q>

9.2.4 False Negative Evaluation Set

9.2.4.1 Pronoun Class

<m1> We </m1> must not feel proud and boast about <m2> ourselves </m2> <eos> <q>

And <m1> my </m1> <unk> is right <comma> because I am not trying to please <m2> myself </m2> <eos> <q>

These false teachers do whatever <m1> they </m1> want <comma> and they are so proud of <m2> themselves </m2> <eos> <q>

9.2.4.2 Name Class

```
<m1> The State Department </m1> <comma> to its credit <comma> has
   already begun distancing <m2> itself </m2> from Mr. Barre <comma>
   evinced by its decision to publish the Gersony report -LRB- which
   the press has ignored -RRB- <eos> <q>

A member in the House leadership and skilled legislator <comma> <m1>
   Mr. Fazio </m1> nonetheless found <m2> himself </m2> burdened not
   only by California 's needs but by Hurricane Hugo amendments he
   accepted in a vain effort to build support in the panel <eos> <q>

This is a key area of development for the future in which <m1> Taiwan
    </m1> can rely on <m2> itself </m2> and need not be limited by
   what can be <unk> abroad <eos> <q>
```

9.2.4.3 Phrase Class

```
<unk> business education may not be widespread in Taiwan at this time
    <comma> yet <m1> a small number of company bosses </m1> have
   taken the lead <m2> themselves </m2> <punct> taking advanced
   courses and setting an example for management <unk> <eos> <q>

So it 's up to <m1> the Lebanese </m1> to decide between <m2>
   themselves </m2> and not do it by force <eos> <q>

part <num> Some House Democrats are trying to head off an appointment
    by President Bush to the board that oversees the savings <punct>
   and <punct> loan bailout <comma> contending that <m1> the
   prospective nominee </m1> is the head of troubled banks <m2>
   himself </m2> <eos> <q>
```

9.3 JSON Specification for DeepDiver

Key	Type	Description	Example
type	STRING	Object type	”state”
name	STRING	Layer name	”2_lstm_ingate”
set	STRING	Set name	valid
epoch	INT>0	Current epoch	100
iteration	INT>0	Current iteration	74
step	INT>0	Current time step	1
sequence	ARRAY(STRING/INT)	JSON array of sequence tokens	[”foo”, ”bar”]
output	STRING/INT	Output token	”bar”
target	STRING/INT	Target token	”bar”
state	ARRAY(FLOAT)	Array of neural activations	[0.079, 0.111]
err	FLOAT	Error of step or iteration	0.00123
input	STRING/INT	Input token	”foo”
success	STRING	”true” or ”false	”true”

Table 9.1: Specification of network state as JSON object for *DeepDiver*. The entries below the line are redundant and thus optional, but can be included to simplify filtering.

DeepDiver is a tool for parsing neural network logs and creating visualizations, such as heatmaps and scatter plots. The parser accepts a newline-separated list of JSON-formatted objects. Each object represents a state of the network, as specified in Table 9.1.

This specification can be applied to both language models and sequence classifiers (many-to-one). Depending on the mode type, keys such as *output*, *target*, and *err* refer to the current step or the current iteration.

The code is available at
https://github.com/kaumanns/DeepDiver

List of Figures

3.1 Schematic comparison of a simple Recurrent Neural Network (RNN) unit and Long Short-Term Memory (LSTM) unit (without the optional peepholes). The upstream layer (blue) feeds into the hidden state (red), and optionally further downstream (purple). Recurrent connections are marked with a dotted arrow. In a simple RNN, the hidden layer is directly connected to itself. In an LSTM unit, the hidden layer does not directly control itself. Instead, it is governed by external units with their own recurrent connections. 18

3.2 A schematic visualization of the connections within an LSTM unit over two time steps. The labels correspond to the layer names defined in Section 3.2.3. Color-filled nodes represent layers: blue is upstream layer, purple is block input, red is memory cells, green is gates, black is block output. White nodes with a colored border indicate an intermediate activation step (red border) or a pointwise multiplication by an adjacent gate (green border). Arrows represent weighted connections of which dotted arrows are recurrent. Arrows that converge to the same node indicate a summation of the weighted input at that node. σ denotes sigmoid activation, g and h denote *tanh* activation. 20

3.3 Schematic visualization of a bidirectional RNN. The upstream layer (blue) feeds into the hidden state (red), and optionally further downstream (purple). 33

5.1 Visualization of the information flow in a recurrent many-to-one sequence classifier. Each arrow represents a full connection between the layers. Recurrent connections are marked by dotted arrows. The task is to predict the antecedent to the anaphor after the entire sequence has been read, including the query token <q>. 49

5.2 Accuracies for LSTM size 1 across different split ratios r of training set size to validation set size 53

5.3 Accuracies for LSTM size 2 across different split ratios r of training set size to validation set size 55

5.4 Accuracies for LSTM size 3 across different split ratios r of training set size to validation set size 56

5.5 Accuracies for LSTM size 4 across different split ratios r of training set size to validation set size 57

5.6 Heatmaps of ***output gate*** states of ***feminine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. Note that in a sequence classifier, *output gates* feed only the recurrent connections, except for the last step where they carry the memory cell activations to the prediction layer. The patterns of permeability are mostly similar to those for the other *genders*. For selected noise words, as well as around the *animals*, *output gate*^{1} seems to selectively shut down completely, possibly in order to communicate information about their relevance to posterior layers. . . 63

5.7 Heatmaps of ***output gate*** states of ***masculine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. Note that in a sequence classifier, *output gates* feed only the recurrent connections, except for the last step where they carry the memory cell activations to the prediction layer. The patterns of permeability are mostly similar to those for the other *genders*. For selected noise words, as well as around the *animals*, *output gate*^{1} seems to selectively shut down completely, possibly in order to communicate information about their relevance to posterior layers. In samples related to *John* (Figure 5.7e, Figure 5.7k), *output gate*^{2} mysteriously shuts down from the *animal* on. 64

5.8 Heatmaps of ***output gate*** states of ***neuter*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. Note that in a sequence classifier, *output gates* feed only the recurrent connections, except for the last step where they carry the memory cell activations to the prediction layer. The patterns of permeability are mostly similar to those for the other *genders*. For selected noise words, as well as around the *animals*, *output gate*^{1} seems to selectively shut down completely, possibly in order to communicate information about their relevance to posterior layers. . . 65

5.9 Heatmaps of ***input gate*** states of ***feminine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. It seems that activation patterns are shared among instances of the same name, which we find mysterious. The expected behavior for the *input gates*, namely shutting out all noise words, does not seem to be implemented. Only *input gate*^{1} tends toward this direction of policy. 66

5.10 Heatmaps of ***input gate*** states of ***masculine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. It seems that activation patterns are shared among instances of the same name, which we find mysterious. The expected behavior for the *input gates*, namely shutting out all noise words, does not seem to be implemented. Only *input gate*[1] tends toward this direction of policy. 67

5.11 Heatmaps of ***input gate*** states of ***neuter*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. It seems that activation patterns are shared among instances of the same name, which we find mysterious. The expected behavior for the *input gates*, namely shutting out all noise words, does not seem to be implemented. Only *input gate*[1] tends toward this direction of policy. Only the adverb-verb clauses in Figure 5.11g and Figure 5.11d trigger this expected behavior. 68

5.12 Heatmaps of ***forget gate*** states of ***feminine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. Overall, *forget gates* are consistently conservative (open) with a few highly focused closed states which trigger forgetting specific *memory cells*, possibly to rule out alternatives. Patterns are related to antecedents and anaphora only. *forget gate*[1] is closed after all *feminine* anaphora. There is no specific permeability after *feminine* antecedents. *forget gate*[1] is closed after all *masculine* names. 71

5.13 Heatmaps of ***forget gate*** states of ***masculine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. Overall, *forget gates* are consistently conservative (open) with a few highly focused closed states which trigger forgetting specific *memory cells*, possibly to rule out alternatives. Patterns are related to antecedents and anaphora only. *forget gate*[3] is closed after all *masculine* anaphora. *forget gate*[1] is closed after *masculine* antecedents *Frank* and *John*. 72

5.14 Heatmaps of ***forget gate*** states of ***neuter*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. Overall, *forget gates* are consistently conservative (open) with a few highly focused closed states which trigger forgetting specific *memory cells*, possibly to rule out alternatives. Patterns are related to antecedents and anaphora only. *forget gate*[1] and *forget gate*[2] are closed after all *neuter* anaphora. There is no specific permeability after *neuter* antecedents. 73

5.15 Heatmaps of ***memory cell*** states of ***feminine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. For example, all penultimate *memory cell*^{3} for *feminine* targets are positively excited. *Mary* triggers a mostly negative excitement in *memory cell*^{2} (e.g. Figure 5.15j). *Susie* is less directly traceable. It seems that the weakly positive excitement it triggers in *memory cell*^{2} from the second position jumps to a negative excitement in *memory cell*^{3} as soon as the *animal* candidate comes in (e.g. Figure 5.15e). A similar effect can be observed in Figure 5.17a, Figure 5.17g, and Figure 5.17i. 74

5.16 Heatmaps of ***memory cell*** states of ***masculine*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. For example, all penultimate *memory cell*^{3} for *masculine* targets are negatively excited. *Peter* triggers a persistent positive excitement in *memory cell*^{1} (e.g. Figure 5.16c). *John* triggers a persistent negative excitement in *memory cell*^{1} which is canceled out when a same-*gender* antecedent follows (Figure 5.16j, Figure 5.16l). 75

5.17 Heatmaps of ***memory cell*** states of ***neuter*** samples in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. For example, all penultimate *memory cell*^{2} for *neuter* targets are positively excited. A positive penultimate *memory cell*^{3} is related to the *cat* target. A negative penultimate *memory cell*^{3} is related to the *dog* target. 76

5.18 Heatmaps of *memory cell* states in a perfect LSTM model with 3 hidden neurons and a 9/1 train/test split. This figure contains the subset of samples that shows that the network does not learn an explicit *order* feature in order to resolve same-*gender* samples. Rather, it learns to overwrite its *memory cells* when it receives a second instance of the same *gender*. The labels indicate which names are preserved in the subsequent *memory cell* patterns ("+" for preserved, "-" for canceled out). Note how in same-*gender* constellations the second name completely cancels out the first name. 77

6.1 Preprocessing pipeline for CoNLL-2012, from raw data to samples with mixed positive and negative mention-pair annotations. 85

6.2 Schematic topologies of the four architectures for the recurrent binary sequence classifier. The input layer denotes a 1-of-k encoded input to the recurrent network. The output layer is shown as a 2-dimensional vector. White downward arrows indicate forward components, white upward arrows indicate backward components of bidirectional units. 93

6.3 Accuracies for learning rate = 0.001 and decay = 1 (no decay) 95

6.4 Accuracies for learning rate = 0.001 and decay = 0.97 96

7.1 Topology of deep bidirectional LSTM binary classifier whose states are analyzed in this chapter. Each rectangle represents an LSTM unit, comprising four layers for which states are saved in the log: *input gate* (i), *forget gate* (f), *output gate* (o), and *memory cell* (c). 115

7.2 Schemes for the policies and saturations of each gate. The labels denote the general regions to which a gate of that policy would tend. 120

7.3 Gate saturations at all tokens across unfiltered test set (10500 network states with 128 neurons per layer). Each point represents a gate neuron. The position on the x-axis indicates the fraction of times it spends right-saturated (> 0.9). The position on the y-axis indicates the fraction of times it spends left-saturated (< 0.1). The grey diagonal indicates where the fractions would add up to one. 121

7.4 Gate saturations at *<m1>* across unfiltered test set (293 network states with 128 neurons per layer). Each point represents a gate neuron. The position on the x-axis indicates the fraction of times it spends right-saturated (> 0.9). The position on the y-axis indicates the fraction of times it spends left-saturated (< 0.1). The grey diagonal indicates where the fractions would add up to one. 122

7.5 Gate saturations at *</m1>* across unfiltered test set (293 network states with 128 neurons per layer). Each point represents a gate neuron. The position on the x-axis indicates the fraction of times it spends right-saturated (> 0.9). The position on the y-axis indicates the fraction of times it spends left-saturated (< 0.1). The grey diagonal indicates where the fractions would add up to one. 124

7.6 Gate saturations at *<m2>* across unfiltered test set (293 network states with 128 neurons per layer). Each point represents a gate neuron. The position on the x-axis indicates the fraction of times it spends right-saturated (> 0.9). The position on the y-axis indicates the fraction of times it spends left-saturated (< 0.1). The grey diagonal indicates where the fractions would add up to one. 125

7.7 Gate saturations at *</m2>* across unfiltered test set (293 network states with 128 neurons per layer). Each point represents a gate neuron. The position on the x-axis indicates the fraction of times it spends right-saturated (> 0.9). The position on the y-axis indicates the fraction of times it spends left-saturated (< 0.1). The grey diagonal indicates where the fractions would add up to one. 126

7.8 Gate saturations at $<q>$ across unfiltered test set (293 network states with 128 neurons per layer). Each point represents a gate neuron. The position on the x-axis indicates the fraction of times it spends right-saturated ($>$ 0.9). The position on the y-axis indicates the fraction of times it spends left-saturated ($<$ 0.1). The grey diagonal indicates where the fractions would add up to one. 127
7.9 States of *memory cell* layers at **all tokens** from the test set, labeled by **output**. Each data point represents a sample, represented by three concatenated *memory cell* layer states and reduced to two dimensions via t-SNE. Each data point represents a layer state, reduced to two dimensions via t-SNE. 130
7.10 States of *memory cell* layers at **anchor tokens** from the **unfiltered** test set, labeled by **output**. Each data point represents a sample, represented by three concatenated *memory cell* layer states and reduced to two dimensions via t-SNE. 132
7.11 States of *memory cell* layers at **anchor tokens** from the **unfiltered** test set, labeled by **target**. Each data point represents a sample, represented by three concatenated *memory cell* layer states and reduced to two dimensions via t-SNE. 133
7.12 States of *memory cell* layers at **anchor tokens** from the ***filtered*** test set, labeled by **output**. Each data point represents a sample, represented by three concatenated *memory cell* layer states and reduced to two dimensions via t-SNE. 134
7.13 States of *memory cell* layers at **anchor tokens** from the ***filtered*** test set, labeled by **nested**. Each data point represents a sample, represented by three concatenated *memory cell* layer states and reduced to two dimensions via t-SNE. 136

List of Tables

3.1 Definitions of layer names. 21

5.1 Number of samples in training set and test set for different split ratios. . . 52
5.2 Accuracies for combinations of split ratio of training set and validation set (train/valid), as well as number of hidden neurons (hidden). 52
5.3 Policies of gate types with respect to their states. Note that the policies of the *forget gate* are flipped compared to the other gates. 60

6.1 Summary of orthographic normalizations for CoNLL-2012. 86
6.2 Context counts 87
6.3 Statistics on sets of different stages of CoNLL-2012 preprocessing (o/w = of which). 91
6.4 Number of parameters for different network layouts. The hidden layer sizes are carefully chosen such that the total number of hidden nodes stays about the same. 92
6.5 Accuracies on the subset of CoNLL-2012 reduced to reflexive mentions. The p-values indicate the statistical significances of the results with respect to a random 50% prediction, as given by the binominal test. Best test set performances (in bold) are highly significant, as achieved with deep bidirectional setups and a slow conservative learning rate (i.e. no decay). 94
6.6 Examples of mentions for the **pronoun** class across the four evaluation subsets. Note the high degree of overlap between the subsets, hinting to confusing context that prevents proper classification. 101
6.7 Examples of mentions for the **names** class across the four evaluation subsets. 102
6.8 Examples of mentions for the **phrases** class across the four evaluation subsets. 103
6.9 Mean cross entropy errors and error variances for all four evaluation subsets. 104

6.10 Performances of the three mention classes for each evaluation subset. Significant deviations from a random 25% baseline distribution across the four evaluation subsets ($p < 0.05$) are marked in bold (measured via binominal test). 104
6.11 Accuracies of each mention class in the test set of a dual-layer bidirectional LSTM classifier. All accuracies are highly significant ($p < 0.005$) with respect to a 50% random classification (measured via binominal test). Mention pairs of class *name* are hardest to classify correctly. Mention pairs of class *pronoun* are easiest to classify. 105

9.1 Specification of network state as JSON object for *DeepDiver*. The entries below the line are redundant and thus optional, but can be included to simplify filtering. 150

References

D. Bahdanau, K. Cho, and Y. Bengio: 2014, Neural machine translation by jointly learning to align and translate. *CoRR*.

P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri: 1999, Exploiting the past and the future in protein secondary structure prediction. *Bioinformatics*, 15(11), pp. 937–946.

Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin: 2003, A neural probabilistic language model. *Journal of Machine Learning Research*, 3, pp. 1137–1155.

Y. Bengio, P. Simard, and P. Frasconi: 1994, Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2), pp. 157–166.

E. Bengtson and D. Roth: 2008, Understanding the value of features for coreference resolution. In *EMNLP 2008*, pp. 294–303.

M. Berglund, T. Raiko, M. Honkala, L. Kärkkäinen, A. Vetek, and J. Karhunen: 2015, Bidirectional recurrent neural networks as generative models-reconstructing gaps in time series. *CoRR*.

T. M. Breuel: 2015a, On the convergence of SGD training of neural networks. *CoRR*.

T. M. Breuel: 2015b, The effects of hyperparameters on SGD training of neural networks. *CoRR*.

T. M. Breuel, A. Ul-Hasan, M. A. Al-Azawi, and F. Shafait: 2013, High-performance OCR for printed english and fraktur using LSTM networks. In *ICDAR 2013*, pp. 683–687.

X. Cheng and R. Voigt: 2015, A deep architecture for coreference resolution. Technical report, Stanford University, Course CS224d: Deep Learning for Natural Language Processing.

K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio: 2014, On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*.

M. H. Christiansen and N. Chater: 1999, Toward a connectionist model of recursion in

human linguistic performance. *Cognitive Science*, 23(2), pp. 157–205.

J. Chung, Çaglar Gülçehre, K. Cho, and Y. Bengio: 2014, Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*.

J. Chung, Çaglar Gülçehre, K. Cho, and Y. Bengio: 2015, Gated feedback recurrent neural networks. *CoRR*.

J. H. Clark and J. P. González-Brenes: 2008, Coreference resolution: Current trends and future directions. *Language and Statistics II Literature Review*, pp. 1–14.

K. Clark: 2015, Neural coreference resolution. Technical report, Stanford University, Course CS224d: Deep Learning for Natural Language Processing.

R. Collobert and J. Weston: 2008, A unified architecture for natural language processing: Deep neural networks with multitask learning. pp. 160–167.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa: 2011, Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12, pp. 2493–2537.

D. Connolly, J. D. Burger, and D. S. Day: 1997, A machine learning approach to anaphoric reference. In *NeMLaP 1994*, pp. 133–144.

G. Cybenko: 1989, Approximation by superpositions of a sigmoidal function. *MCSS 1989*, 2(4), pp. 303–314.

Y. N. Dauphin, H. de Vries, J. Chung, and Y. Bengio: 2015, RMSProp and equilibrated adaptive learning rates for non-convex optimization. *CoRR*.

J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng: 2012, Large scale distributed deep networks. In *NIPS 2012*, pp. 1232–1240.

K. Doya: 1995, Supervised learning in recurrent networks. *Handbook of brain theory and neural networks.*

J. Duchi, E. Hazan, and Y. Singer: 2011, Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12, pp. 2121–2159.

G. Durrett and D. Klein: 2013, Easy victories and uphill battles in coreference resolution. In *EMNLP 2013*, pp. 1971–1982.

J. L. Elman: 1990, Finding structure in time. *Cognitive Science*, 14(2), pp. 179–211.

J. L. Elman: 1991, Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7(2-3), pp. 195–225.

D. Erhan, Y. Bengio, A. C. Courville, P. Manzagol, P. Vincent, and S. Bengio: 2010,

Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11, pp. 625–660.

R. P. Feynman: 1998, Cargo cult science. In *The Art and Science of Analog Circuit Design*, EDN series for design engineers, chapter 6, pp. 55–61. Newnes, Woburn.

J. A. Fodor and Z. W. Pylyshyn: 1988, Connectionism and cognitive architecture: A critical analysis. In *Connections and symbols (cognition special issue)*, pp. 3–71. MIT Press, Cambridge, MA, USA.

V. Frinken, A. Fornés, J. Lladós, and J. Ogier: 2012a, Bidirectional language model for handwriting recognition. In *IAPR-SSPR&SPR 2012*, pp. 611–619.

V. Frinken, F. Zamora-Martínez, S. E. Boquera, M. J. C. Bleda, A. Fischer, and H. Bunke: 2012b, Long-short term memory neural networks language modeling for handwriting recognition. In *ICPR 2012*, pp. 701–704.

J. Garson: 2015, Connectionism. In *The stanford encyclopedia of philosophy*. http://plato.stanford.edu/archives/spr2015/entries/connectionism/.

F. A. Gers, J. Schmidhuber, and F. A. Cummins: 2000, Learning to forget: Continual prediction with LSTM. *Neural computation*, 12(10), pp. 2451–2471.

F. A. Gers, N. N. Schraudolph, and J. Schmidhuber: 2002, Learning precise timing with LSTM recurrent networks. *Journal of Machine Learning Research*, 3, pp. 115–143.

C. L. Giles and T. Maxwell: 1987, Learning, invariance, and generalization in high-order neural networks. *Applied Optics*, 26(23), pp. 4972–4978.

C. L. Giles, C. B. Miller, D. Chen, H. Chen, G. Sun, and Y. Lee: 1992, Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3), pp. 393–405.

X. Glorot and Y. Bengio: 2010, Understanding the difficulty of training deep feedforward neural networks. In *AISTATS 2010*, pp. 249–256.

Y. Goldberg: 2015, A primer on neural network models for natural language processing. *CoRR*.

M. W. Goudreau, C. L. Giles, S. T. Chakradhar, and D. Chen: 1994, First-order versus second-order single-layer recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(3), pp. 511–513.

A. Graves: 2012, *Supervised sequence labelling with recurrent neural networks*, PhD thesis, University of Munich.

A. Graves and N. Jaitly: 2014, Towards end-to-end speech recognition with recurrent

neural networks. In *ICML 2014*, pp. 1764–1772.

A. Graves and J. Schmidhuber: 2008, Offline handwriting recognition with multidimensional recurrent neural networks. In *NIPS 2008*, pp. 545–552.

A. Graves, D. Eck, N. Beringer, and J. Schmidhuber: 2004, Biologically plausible speech recognition with LSTM neural nets. In *Bio-ADIT*, pp. 127–136.

A. Graves, N. Jaitly, and A. Mohamed: 2013a, Hybrid speech recognition with deep bidirectional LSTM. In *ASRU 2013*, pp. 273–278.

A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber: 2009, A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5), pp. 855–868.

A. Graves, A. Mohamed, and G. E. Hinton: 2013b, Speech recognition with deep recurrent neural networks. In *ICASSP 2013*, pp. 6645–6649.

A. Graves, G. Wayne, and I. Danihelka: 2014, Neural turing machines. *CoRR*.

E. Grefenstette, K. M. Hermann, M. Suleyman, and P. Blunsom: 2015, Learning to transduce with unbounded memory. In *NIPS 2015*, pp. 1828–1836.

K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber: 2015, LSTM: A search space odyssey. *CoRR*.

A. Haghighi and D. Klein: 2007, Unsupervised coreference resolution in a nonparametric bayesian model. In *ACL 2007*, pp. 848.

A. Haghighi and D. Klein: 2009, Simple coreference resolution with rich syntactic and semantic features. In *EMNLP 2009*, pp. 1152–1161.

A. Haghighi and D. Klein: 2010, Coreference resolution in a modular, entity-centered model. In *ACL 2010*, pp. 385–393.

A. Halevy, P. Norvig, and F. Pereira: 2009, The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2), pp. 8–12.

J. Hammerton: 2003, Named entity recognition with long short-term memory. In *CoNLL 2003*, pp. 172–175.

K. M. Hermann, T. Kociský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom: 2015, Teaching machines to read and comprehend. In *NIPS 2015*, pp. 1693–1701.

S. Hochreiter and J. Schmidhuber: 1997, Long short-term memory. *Neural Computation*, 9(8), pp. 1735–1780.

S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber: 2001, Gradient flow in recur-

rent nets: The difficulty of learning long-term dependencies. In *A field guide to dynamical recurrent neural networks*. IEEE Press.

J. J. Hopfield: 1982, Neural networks and physical systems with emergent collective computational abilities. *National Academy of Sciences*, 79(8), pp. 2554–2558.

K. Hornik: 1991, Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2), pp. 251–257.

K. Hornik, M. B. Stinchcombe, and H. White: 1989, Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5), pp. 359–366.

P. A. Jansen and S. Watter: 2012, Strong systematicity through sensorimotor conceptual grounding: An unsupervised, developmental approach to connectionist sentence processing. *Connection Science*, 24(1), pp. 25–55.

R. Józefowicz, W. Zaremba, and I. Sutskever: 2015, An empirical exploration of recurrent network architectures. In *ICML 2015*, pp. 2342–2350.

A. Karpathy, J. Johnson, and F. Li: 2015, Visualizing and understanding recurrent networks. *CoRR*.

A. Kehler and H. Rohde: 2013, A probabilistic reconciliation of coherence-driven and centering-driven theories of pronoun interpretation. *Theoretical Linguistics*, 39(1-2), pp. 1–37.

D. Khashabi: 2013, On the recursive neural networks for relation extraction and entity recognition. Technical report, UIUC.

D. P. Kingma and J. Ba: 2014, Adam: A method for stochastic optimization. *CoRR*.

H. Kobdani: 2012, *A modular framework for coreference resolution*, PhD thesis, University of Stuttgart.

H. Kobdani, H. Schütze, M. Schiehlen, and H. Kamp: 2011, Bootstrapping coreference resolution using word associations. In *ACL 2011*, pp. 783–792.

J. Koutník, K. Greff, F. J. Gomez, and J. Schmidhuber: 2014, A clockwork RNN. In *ICML 2014*, pp. 1863–1871.

A. Kumar, O. Irsoy, J. Su, J. Bradbury, R. English, B. Pierce, P. Ondruska, I. Gulrajani, and R. Socher: 2015, Ask me anything: Dynamic memory networks for natural language processing. *ICML 2016*, pp. 1378–1387.

D. Küçük and M. T. Yöndem: 2015, A knowledge-poor pronoun resolution system for turkish. *CoRR*.

H. Lee, A. Chang, Y. Peirsman, N. Chambers, M. Surdeanu, and D. Jurafsky: 2013,

Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational Linguistics*, 39(4), pp. 885–916.

H. Lee, Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky: 2011, Stanford's multi-pass sieve coreference resolution system at the CoNLL-2011 shared task. In *CoNLL 2011*, pp. 28–34.

N. Léonard, S. Waghmare, Y. Wang, and J. Kim: 2015, rnn: Recurrent library for Torch. *CoRR*.

J. Lighthill: 1973, Artificial intelligence: A paper symposium. In chapter Artificial Intelligence: A General Survey. Science Research Council.

A. L. Maas, A. Y. Hannun, D. Jurafsky, and A. Y. Ng: 2014, First-pass large vocabulary continuous speech recognition using bi-directional recurrent DNNs. *CoRR*.

C. D. Manning: 2015, Computational linguistics and deep learning. *Computational Linguistics*, pp. 701–707.

J. Martens and I. Sutskever: 2011, Learning recurrent neural networks with hessian-free optimization. In *ICML 2011*, pp. 1033–1040.

D. A. Medler: 1998, A brief history of connectionism. *Neural Computing Surveys*, 1, pp. 61–101.

T. Mikolov, K. Chen, G. Corrado, and J. Dean: 2013a, Efficient estimation of word representations in vector space. *CoRR*.

T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur: 2010, Recurrent neural network based language model. In *Interspeech 2010*, pp. 1045–1048.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean: 2013b, Distributed representations of words and phrases and their compositionality. In *NIPS 2013*, pp. 3111–3119.

M. Minsky and S. Papert: 1969, *Perceptron: An introduction to computational geometry*. MIT Press, Cambridge, MA, USA.

T. M. Mitchell: 1997, *Machine learning*. McGraw-Hill, New York, NY, USA.

R. Mitkov: 1999, Anaphora resolution: The state of the art. Technical report, University of Wolverhampton.

D. Monner and J. A. Reggia: 2012, A generalized LSTM-like training algorithm for second-order recurrent neural networks. *Neural Networks*, 25, pp. 70–83.

G. Montavon, S. Bach, A. Binder, W. Samek, and K.-R. Müller: 2015, Explaining non-

linear classification decisions with deep Taylor decomposition. *CoRR*.

K. Moreland: 2009, Diverging color maps for scientific visualization (expanded). In *ISVC 2009*, pp. 92–103.

W. C. Morris, G. W. Cottrell, and J. Elman: 1998, A connectionist simulation of the empirical acquisition of grammatical relations. In *Hybrid neural systems, revised papers from a workshop*, pp. 175–193. Springer, London.

A. Newell and H. A. Simon: 1976, Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19(3), pp. 113–126.

V. Ng: 2008, Unsupervised models for coreference resolution. In *EMNLP 2008*, pp. 640–649.

V. Ng: 2010, Supervised noun phrase coreference research: The first fifteen years. In *ACL 2010*, pp. 1396–1411.

V. Ng and C. Cardie: 2002a, Combining sample selection and error-driven pruning for machine learning of coreference rules. In *EMNLP 2002*, pp. 55–62.

V. Ng and C. Cardie: 2002b, Improving machine learning approaches to coreference resolution. In *ACL 2002*, pp. 104–111.

F. Olsson: 2004, A survey of machine learning for reference resolution in textual discourse. Technical report, SICS Swedish ICT.

C. Orasan and R. J. Evans: 2011, NP animacy identification for anaphora resolution. *CoRR*.

H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. K. Ward: 2015, Deep sentence embedding using the long short term memory network: Analysis and application to information retrieval. *CoRR*.

R. Pascanu, T. Mikolov, and Y. Bengio: 2013, On the difficulty of training recurrent neural networks. In *ICML 2013*, pp. 1310–1318.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay: 2012, Scikit-learn: Machine learning in Python. *CoRR*.

J. Pennington, R. Socher, and C. D. Manning: 2014, Glove: Global vectors for word representation. In *EMNLP 2014*, pp. 1532–1543.

J. B. Pollack: 1988, Recursive auto-associative memory: Devising compositional distributed representations. Technical report, Computing Research Laboratory, New Mexico

State University, Las Cruces.

H. Poon and P. M. Domingos: 2008, Joint unsupervised coreference resolution with Markov logic. In *EMNLP 2008*, pp. 650–659.

S. Pradhan, A. Moschitti, N. Xue, O. Uryupina, and Y. Zhang: 2012, CoNLL-2012 Shared task: Modeling multilingual unrestricted coreference in OntoNotes. In *EMNLP-CoNLL 2012*, pp. 1–40.

R. Q. Quiroga, G. Kreiman, C. Koch, and I. Fried: 2008, Sparse but not 'grandmother-cell' coding in the medial temporal lobe. *Trends in cognitive sciences*, 12(3), pp. 87–91.

K. Raghunathan: 2010, *Simple coreference resolution with rich syntactic and semantic features: Is it enough?*, Master's thesis, Stanford University.

K. Raghunathan, H. Lee, S. Rangarajan, N. Chambers, M. Surdeanu, D. Jurafsky, and C. Manning: 2010, A multi-pass sieve for coreference resolution. In *EMNLP 2010*, pp. 492–501.

A. Rahman and V. Ng: 2014, Narrowing the modeling gap: A cluster-ranking approach to coreference resolution. *CoRR*.

M. Richardson, C. J. C. Burges, and E. Renshaw: 2013, MCTest: A challenge dataset for the open-domain machine comprehension of text. In *EMNLP 2013*, pp. 193–203.

M. Riedmiller and H. Braun: 1993, A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *ICNN 1993*, pp. 586–591.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams: 1988, Neurocomputing: Foundations of research. In chapter Learning internal representations by error propagation, pp. 673–695. MIT Press, Cambridge, MA, USA.

D. E. Rumelhart, J. L. McClelland, and PDP Research Group: 1986, *Parallel distributed processing: Explorations in the microstructure of cognition*. MIT Press, Cambridge, MA, USA.

H. Sak, A. Senior, and F. Beaufays: 2014, Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *CoRR*.

W. Samek, A. Binder, G. Montavon, S. Bach, and K.-R. Müller: 2015, Evaluating the visualization of what a deep neural network has learned. *CoRR*.

M. Schuster: 1999, *On supervised learning from sequential data with applications for speech recognition*, PhD thesis, Nara Institute of Science and Technology.

M. Schuster and K. K. Paliwal: 1997, Bidirectional recurrent neural networks. *IEEE*

Transactions on Signal Processing, 45(11), pp. 2673–2681.

H. Schütze: 1993, Part-of-speech induction from scratch. In *ACL 1993*, pp. 251–258.

H. Schwenk and J.-L. Gauvain: 2005, Training neural network language models on very large corpora. In *EMNLP-HLT 2005*, pp. 201–208.

H. T. Siegelmann and E. D. Sontag: 1995, On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1), pp. 132–150.

W. M. Soon, H. T. Ng, and D. C. Y. Lim: 2001, A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27(4), pp. 521–544.

V. Stoyanov, N. Gilbert, C. Cardie, and E. Riloff: 2009, Conundrums in noun phrase coreference resolution: Making sense of the state-of-the-art. In *ACL-AFNLP 2009*, pp. 656–664.

R. Stuckardt: 2007, Applying backpropagation networks to anaphor resolution. In *DAARC 2007*, pp. 107–124.

S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus: 2015, End-to-end memory networks. *NIPS 2015*, pp. 2440–2448.

M. Sundermeyer, T. Alkhouli, J. Wuebker, and H. Ney: 2014, Translation modeling with bidirectional recurrent neural networks. In *EMNLP 2014*, pp. 14–25.

M. Sundermeyer, I. Oparin, J. Gauvain, B. Freiberg, R. Schlüter, and H. Ney: 2013, Comparison of feedforward and recurrent neural network language models. In *ICASSP 2013*, pp. 8430–8434.

M. Sundermeyer, R. Schlüter, and H. Ney: 2012, LSTM neural networks for language modeling. In *ISCA 2012*, pp. 194–197.

I. Sutskever, J. Martens, and G. E. Hinton: 2011, Generating text with recurrent neural networks. In *ICML 2011*, pp. 1017–1024.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton: 2013, On the importance of initialization and momentum in deep learning. In *ICML 2013*, pp. 1139–1147.

W. Tabor: 1994, *Syntactic innovation: A connectionist model*, PhD thesis, Stanford University.

T. Tieleman and G. Hinton: 2012, Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4, pp. 2.

O. Uryupina and A. Moschitti: 2015, A state-of-the-art mention-pair model for coreference

resolution. In *Lexical and computational semantics*, pp. 289–298.

K. Van Deemter and R. Kibble: 2000, On coreferring: Coreference in MUC and related annotation schemes. *Computational linguistics*, 26(4), pp. 629–637.

L. Van der Maaten and G. Hinton: 2008, Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(2579-2605), pp. 85.

M. Wand, J. Koutník, and J. Schmidhuber: 2016, Lipreading with long short-term memory. In *ICASSP 2016*, pp. 6115–6119.

C. Weng, D. Yu, S. Watanabe, and B. F. Juang: 2014, Recurrent deep neural networks for robust speech recognition. In *ICASSP 2014*, pp. 5532–5536.

P. J. Werbos: 1988, Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4), pp. 339–356.

J. Weston, A. Bordes, S. Chopra, and T. Mikolov: 2015, Towards AI-complete question answering: A set of prerequisite toy tasks. *CoRR*.

J. Weston, S. Chopra, and A. Bordes: 2014, Memory networks. *CoRR*.

R. J. Williams and D. Zipser: 1989, A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2), pp. 270–280.

R. J. Williams and D. Zipser: 1995, Gradient-based learning algorithms for recurrent networks and their computational complexity. *Back-propagation: Theory, architectures and applications*, pp. 433–486.

W. Williams, N. Prasad, D. Mrva, T. Ash, and T. Robinson: 2015, Scaling recurrent neural network language models. In *ICASSP 2015*, pp. 5391–5395.

Z. Wu and S. King: 2016, Investigating gated recurrent networks for speech synthesis. In *ICASSP 2016*, pp. 5140–5144.

K. Yao, T. Cohn, K. Vylomova, K. Duh, and C. Dyer: 2015, Depth-gated recurrent neural networks. *CoRR*.

M. D. Zeiler: 2012, ADADELTA: An adaptive learning rate method. *CoRR*.

D. Zhekova: 2013, *Towards multilingual coreference resolution*, PhD thesis, University of Bremen.

D. Zhekova and S. Kübler: 2013, Machine learning for mention head detection in multilingual coreference resolution. In *RANLP 2013*, pp. 747–754.