Enhancing Representation Learning with Tensor Decompositions for Knowledge Graphs and High Dimensional Sequence Modeling

Yinchong Yang

München 2017

Enhancing Representation Learning with Tensor Decompositions for Knowledge Graphs and High Dimensional Sequence Modeling

Yinchong Yang

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Yinchong Yang
aus Dalian, Liaoning, China VR

München, den 20.12.2017

Erstgutachter: Prof. Dr. Volker Tresp

Zweitgutachter: Prof. Dr. Gunnar Rätsch

Drittgutachter: Prof. Dr. Bertram Müller-Myhsok

Tag der mündlichen Prüfung: 27.03.2018

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Yang, Yinchong

Name, Vorname

München, 17.04.2018

Ort, Datum

Unterschrift Doktorand/in

Formular 3.2

Contents

Abstract

The capability of processing and digesting raw data is one of the key features of a human-like artificial intelligence system. For instance, real-time machine translation should be able to process and understand spoken natural language, and autonomous driving relies on the comprehension of visual inputs. Representation learning is a class of machine learning techniques that autonomously learn to derive latent features from raw data. These new features are expected to represent the data instances in a vector space that facilitates the machine learning task. This thesis studies two specific data situations that require efficient representation learning: knowledge graph data and high dimensional sequences.

In the first part of this thesis, we first review multiple relational learning models based on tensor decomposition for knowledge graphs. We point out that relational learning is in fact a means of learning representations through one-hot mapping of entities. Furthermore, we generalize this mapping function to consume a feature vector that encodes all known facts about each entity. It enables the relational model to derive the latent representation instantly for a new entity, without having to re-train the tensor decomposition.

In the second part, we focus on learning representations from high dimensional sequential data. Sequential data often pose the challenge that they are of variable lengths. Electronic health records, for instance, could consist of clinical event data that have been collected at subsequent time steps. But each patient may have a medical history of variable length. We apply recurrent neural networks to produce fixed-size latent representations from the raw feature sequences of various lengths. By exposing a prediction model to these learned representations instead of the raw features, we can predict the therapy prescriptions more accurately as a means of clinical decision support. We further propose Tensor-Train recurrent neural networks. We give a detailed introduction to the technique of tensorizing and decomposing large weight matrices into a few smaller tensors. We demonstrate the specific algorithms to perform the forward-pass and the back-propagation in this setting. Then we apply this approach to the input-to-hidden weight matrix in recurrent neural networks. This novel architecture can process extremely high dimensional sequential features such as video data. The model also provides a promising solution to processing sequential features with high sparsity. This is, for instance, the case with electronic health records, since they are often of categorical nature and have to be binary-coded. We incorporate a statistical survival model with this representation learning model, which shows superior prediction quality.

Zusammenfassung

Repräsentations-Lernen ist ein Teilgebiet des maschinellen Lernens, welches sich mit dem autonomen Erlernen von latenten Attributen aus den Rohdaten befasst. Es wird davon ausgegangen, dass diese neu erlernten Attribute die Datenpunkte in einem neuen Vektorraum repräsentieren, in welchen die eigentliche Aufgabe des maschinellen Lernens leichter zu lösen ist. Die Fähigkeit, rohe Daten-Attribute verarbeiten zu können, gehört zu den Kerneigenschaften einer menschenähnlichen künstlichen Intelligenz. Maschinelle Übersetzung in Echtzeit basiert beispielsweise auf dem maschinellen Verständnis gesprochener Sprache, wohingegen autonomes Fahren teilweise auf visuellen Eingaben basiert. Das effiziente Erlernen von guten Repräsentationen ist dadurch möglich, dass sich die Kapazität der Datensammlung und Datenspeicherung in den letzten Jahren enorm verbessert hat. Einerseits erlaubt das zunehmende Datenvolumen das Training von größeren und ausdrucksstärkeren Modellen des maschinellen Lernens. Andererseits führen Daten in anwachsender Vielfalt auch dazu, Methoden des Repräsentations-Lernens für spezifische Datenformate zu entwickeln. Diese Arbeit beschäftigt sich mit zwei solchen speziellen Datensituationen, welche nach effizientem Repräsentations-Lernen verlangen: Wissensgraphen-Daten und hochdimensionale sequenzielle Daten.

Im ersten Teil dieser Arbeit geben wir einen Überblick über mehrere gängige Modelle basierend auf Tensor-Faktorisierung für das Lernen von Relationen aus Wissensgraphen. Wir zeigen, dass bei diesen Modellen die Relationen dadurch modelliert werden, indem man eine latente Repräsentation für jede Entität durch einen indizierenden One-Hot-Vektor codiert. Diese Codierung lässt sich dann aber erweitern um einen Attributvektor, der alle relationalen Informationen einer Entität beinhaltet. Somit kann man latente Repräsentation für neue Entitäten ableiten, ohne die Faktorisierung neu berechnen zu müssen.

Im zweiten Teil der Arbeit fokussieren wir uns auf das Repräsentations-Lernen aus hochdimensionalen Sequenzen. Sequentielle Daten stellen oft die Herausforderung dar, dass die Sequenzen von unterschiedlicher Länge sein können. Klinische Gesundheitsdaten bestehen beispielsweise aus wiederholten Messungen zu subsequenten Zeitpunkten, aber wie viele Messungen an einem Patienten vorgenommen worden ist, variiert stark vom Patient zu Patient. Wir setzen rekurrente neuronale Netze ein, um einen Repräsentationsvektor von fester Größe aus der Sequenz variabler Länge zu erlernen. Ein darauf basierendes Vorhersage-Modell soll die Therapie-Entscheidungen zum Zweck der klinischen Entscheidungsuntersützung prognostizieren. Des Weiteren entwickeln wir das Tensor-Train

rekurrente neuronale Netzwerk. Wir geben eine detaillierte Einführung und Ableitung der Tensorisierung und Faktorisierung beliebiger Gewichts-Matrizen. Wir stellen auch die Algorithmen sowohl für den Forward-Pass als auch für die Back-Propagation dar. Wir setzen diese Technik ein, um die Input-zu-Hidden Gewichts-Matrix in rekurrenten neuronalen Netzwerken auf mehrere kleinere Tensoren zu komprimieren. Dadurch ist ein rekurrentes neuronales Netzwerk in der Lage, extrem hochdimensionale Sequenzen wie z.B. Video-Daten zu verarbeiten. Auch im Falle von klinischen Daten bietet diese neue Architektur eine Lösung dafür, dass die kategorialen Attribute durch Binärkodierung hohe Sparsität aufweisen. Ein statistisches Überlebensdauer-Model, welches die erlernte Repräsentation als Eingabe konsumiert, zeigt auch verbesserte Vorhersage.

Acknowledgements

This PhD work would not have been possible without the contribution and support of many people.

First of all, my utmost and deepest gratitude goes to my supervisor and mentor, Prof. Dr. Volker Tresp. Volker provided me with an excellent platform to undertake my research at LMU and Siemens AG, and gave me the great opportunity to participate in the BMWi project of Klinische Datenintelligenz. He introduced me to relational learning and the Tensor-Train model, which form the two pillars of this very work. He always encourages me to be innovative and to think big, and never hesitates to share his own insights to any topic I would like to discuss with him. I also deeply appreciate his effort in helping me prepare all the publications. I would like to thank Dr. Ulli Waltinger, head of the research group of Machine Intelligence at Siemens, for providing me with the first-class infrastructure for my research work. And I am really grateful for his offering me the opportunity to work in his group in the future. I am very honored that Prof. Dr. Gunnar Rätsch and Prof. Dr. Bertram Müller-Myhsok agreed to be external examiners of my thesis. I want to thank them for their most constructive and enlightening comments to this work.

More than three years ago, it were Dr. Fabian Scheipl at Institute for Statistics, LMU Munich, and Marco Schreyer at PwC Germany, Stuttgart, who recommended me to further my academic training in pursuing a PhD. Now I do feel that I owe them my thanks for helping me make the right decision.

I feel very fortunate to have been working with my colleagues Dr. Denis Krompaß, Cristóbal Esteban and Stephan Baier in the last three years. Especially at the beginning of my PhD, Denis has been like a second supervisor and mentor to me, giving me advices on a variety of both technical and administrative topics. He was also extremely supportive and helpful during my work on the Tensor-Train RNN paper, by offering me a lot of assistance and suggestions. Cristóbal's outstanding work in machine learning with healthcare data has greatly inspired my own and I would like to thank him for sharing with me his valuable experience. The path to a PhD is not an easy one. It is as important to have these wonderful advisers as to have fellow students, with whom one can exchange information, encouragement and congratulation. I'm grateful that I could work with Stephan in this way for the last years, and many thanks for his reading and comments on the German abstract of this work. I have also thankfully received a lot of help and advice from Dr.

Sigurd Spieckermann and Yi Huang at Siemens.

Last but not least, I would like to thank my parents for being understanding and supportive throughout the years that I spend overseas. Very special thanks go to Juliette for her proofreading and correction of the entire thesis, for her invaluable friendship for almost two decades, and for her introducing me to so many new dimensions in life.

List of Publications and Declaration of Authorship

- Yinchong Yang, Cristóbal Esteban, and Volker Tresp. Embedding mapping approaches for tensor factorization and knowledge graph modelling. In Harald Sack, Eva Blomqvist, Mathieu d'Aquin, Chiara Ghidini, Simone Paolo Ponzetto, and Christoph Lange, editors, *The Semantic Web. Latest Advances and New Domains: 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 – June 2, 2016, Proceedings*, pages 199–213. Springer International Publishing, 2016

  I conceived of the original research contributions and performed all implementations and evaluations. I wrote the initial draft of the manuscript and did most of the subsequent corrections. I regularly discussed this work with the co-author, Cristóbal Esteban, and my supervisor, Volker Tresp, who also assisted me in improving the manuscript.

  This published work serves as Chapter 2.

- Yinchong Yang, Peter A. Fasching, and Volker Tresp. Predictive modeling of therapy decisions in metastatic breast cancer with recurrent neural network encoder and multinomial hierarchical regression decoder. In *Proceedings of the IEEE International Conference on Healthcare Informatics (ICHI)*, Park City, Utah, USA, 23–26 Aug 2017. IEEE

  I conceived of the original research contributions and performed all implementations and evaluations. I wrote the initial draft of the manuscript and did most of the subsequent corrections. I regularly discussed this work with the co-author, Peter A. Fasching, and my supervisor, Volker Tresp, who also assisted me in improving the manuscript.

This published work serves as Chapter 3.

- Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3891–3900, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. JMLR

  I conceived of the original research contributions and performed all implementations and evaluations. I wrote the initial draft of the manuscript and did most of the subsequent corrections. I regularly discussed this work with the co-author, Denis Krompaß, and my supervisor, Volker Tresp, who also assisted me in improving the manuscript.

  This published work serves as Chapter 4.

- Yinchong Yang, Peter A. Fasching, and Volker Tresp. Modeling progression free survival in breast cancer with tensorized recurrent neural networks and accelerated failure time model. In *Machine Learning for Healthcare 2017*, volume 68 of *Proceedings of Machine Learning Research*, Northeastern University, Boston, USA, 18–19 Aug 2017. JMLR

  I conceived of the original research contributions and performed all implementations and evaluations. I wrote the initial draft of the manuscript and did most of the subsequent corrections. I regularly discussed this work with the co-author, Peter A. Fasching, and my supervisor, Volker Tresp, who also assisted me in improving the manuscript.

  This published work serves as Chapter 5.

Chapter 1

Introduction

1.1 Representation Learning

1.1.1 Motivation

One of the ultimate goals of artificial intelligence is the creation of computer agents that can undertake specific tasks, such as prediction and control, which would otherwise require human intelligence [65, 91, 89]. Earlier attempts in this field, such as symbolic approaches [80], involved the representation of human knowledge as theorems and facts and applying logical inference rules. The most successful instances of this class of approaches are probably the expert systems studied during the 70s and 80s [69]. Expert systems turned out difficult to maintain and incapable of autonomous learning from new data [29, 91]. Statistical learning [109] and soft computing [21], on the contrary, have been contributing more and more to the realization of a self learning agent. Instead of consulting human experts and hard-coding their knowledge into a computer system, one solely presents the agent data samples that define the task it is expected to undertake. For example, by processing enough images of handwritten digits and the corresponding labels, an agent learns the mapping pattern between the digit and the label, and can then predict labels for new handwritten digits. That is to say, instead of being programmed to do so, the agent *learns* to induce abstract –though sometimes not necessarily interpretable– rules from the data [16]. These rules are expressed as hierarchies of mathematical operations [10]. The designing and training of such an agent are major tasks in machine learning. It is often initialized with close to no prior knowledge and has to learn from its own errors.

In recent years, the noticeable improvement of computer hardware techniques have

enabled the collection and processing of ever larger amounts of data, which greatly contributed to the remarkable advances in machine learning [71].

Within the field of machine learning, especially the data-intensive and data-driven methods are profiting even more from the improving data techniques. Among these approaches, artificial neural networks are probably the most successful and representative examples [11]. They exploit modern hardware techniques in two directions: on one side, the computation power of modern processing units is evolving. In addition, GPU [97] and TPU [105] are also being applied for more efficient algebraic operations. The modeling power, i.e., the expressiveness of a neural network, is often closely related to its complexity [10]. To this end, the advances in computation power allow for training larger and more complicated neural networks. On the other side, the storage capacity has been booming drastically, enabling the training of neural networks with data samples in ever larger volume. This is often crucial for neural networks, because more complex networks with more parameters require more data samples to train [11]. Furthermore, the growing storage capacity has been encouraging the exploitation of data formats in larger variety. In addition to the structured data in traditional table form, less structured data such as images, voices and videos are now also serving as training data. Obviously, the ability to process and digest these raw data is a key feature for human-like artificial intelligence. For instance, real-time machine translation depends on understanding natural spoken language[102, 25, 24]; autonomous driving is largely based on visual sensory input in vehicles [13, 78, 22]; clinical decision support systems integrate all available information sources of a patient [38, 26, 116], from laboratory analysis to radiology images [23, 70, 112].

These raw data formats, being potentially noisy, high dimensional, sparse and of variable sizes, also pose novel challenges to designing machine learning models. Earlier approaches in handling such raw data rely on consulting domain human knowledge to design and hard code features that are to be extracted from raw data. For instance, one calculates histograms [15] and wavelets from image data [2], optical flow [101, 18] and trajectories [111, 72] from video data. For natural language data, the n-grams are often quite popular choices [17, 57]. The extraction is performed as a pre-processing step and once extracted, these features remain constant to serve as direct input to machine learning models. The hand-designed features enjoy the advantage of being well interpretable, but are sometimes not scalable to larger data volumes or new characteristics in data. They are also prone to over-specification [62] and can in fact never outperform human knowledge. The latest research developments suggest to integrate this pre-processing step into the machine learning

model itself. In other words, a machine learning model should *learn* to extract from the raw data latent but more exploratory features to facilitate its own predictive task, being typically regression and classification [11]. These features are also known as (latent) representations, because they represent the data instances now in a latent space. They are also dynamic instead of fixed during training of the prediction model. Such a mapping from raw features to the latent representation can be realized efficiently by multiple layers of neural networks. They are proven to be universal approximators [49, 48, 30], and they can be trained jointly with the prediction model, applying back propagation in a supervised and end-to-end fashion. In contrast to hand-designed feature extraction, these learned representations are task specific and only in rare cases human interpretable. However, in a wide variety of AI tasks varying from object recognition [28, 90, 32] to playing the Go game [95, 96], these deep neural networks outperform approaches using hand designed features. They are also adaptable to novel characteristics in the data, and they can largely profit from the growing volume and variety of data being collected nowadays.

One could draw an interesting analogy in the development of artificial intelligence: machine learning adapts to improving hardware capacity better than expert systems because the former can learn the reasoning rules –though less interpretable– *automatically* from data samples; while an expert system relies completely on the hard coding of human knowledge and fails to profit from the exploding data volume. In machine learning, latest representation learning approaches focuses on learning latent representations –though less interpretable– *automatically* from raw data, instead of utilizing hand designed representations. In the last decades, the trend of development in AI seems to be decreasing the involvement of human knowledge, and increasing the volume and variety of data that an AI system can access and learn from. In other words, one does not tell an AI agent *how* to carry out a task. Instead, one tells it *what* to do. And with the improving hardware capacity and growing data volume, the AI agent can be expected to figure out *how* by itself.

This work focuses on two specific data situations that heavily rely on efficient representation learning, namely knowledge graph data and high dimensional sequential data. The remainder of this work is structured as follows: In subsection 1.1.3 we first define a generic framework to define machine learning models that consists of autonomous representation learning.

Section 1.2 addresses the task of modeling knowledge graph data. We first introduce the data situation in Subsection 1.2.1 and then review multiple popular relational learning

models in Subsection 1.2.2. In Subsection 1.2.3, we elaborate the relation between these relational learning models and representation learning within the generic framework. Based on this insight, we propose in Subsection 1.2.4 a new relational learning concept to learn the mapping from known facts to the latent representations. Paragraph 1.2.4.1 gives a brief introduction to our published work [114].

In section 1.3 we study the second data situation of high dimensional sequential data. The major challenge as well as some classical solutions are first discussed in Subsection 1.3.1. Then, Subsection 1.3.2 reviews the latest recurrent neural network models, which provide another solution in handling sequential data. Paragraph 1.3.2.1 serves as a brief introduction to our published work of [116], applying recurrent neural networks to process patient history data. Furthermore, we tackle the challenge that sequential data could also be of high dimensionality and sparsity. We propose a novel recurrent neural network architecture integrating Tensor-Train decomposition in Subsection 1.3.4. In Paragraph 1.3.4.1, we give a brief introduction to our published work on applying this model on video data as high dimensional sequences, and in Paragraph 1.3.4.2 we introduce our work of [115], which applies the same model on health data.

The Sections 2, 3, 4 and 5 consist of our published works, respectively.

Section 6 is a summary of our work. It highlights the major contributions and gives our interpretation of the research results.

The appendices include a numerical example to illustrate relational learning and representation mapping . For the Tensor-Train recurrent networks, we provide more technical details such as a forward pass algorithm and the gradients calculation, as well as simulation studies to verify the implementation.

1.1.2 Terminology and Notations

For the rest of the work, a *data instance* refers to what is usually understood as *data point* or a single *data sample* in a typical machine learning setting. It could be, for instance, a single row-vector in a design matrix, a matrix representing a black and white image, a three-way tensor storing an RGB image, and a sequence of words that build a sentence, etc. We use the term *feature* instead of, e.g., *variable* or *predictor*. By our convention, the term recurrent neural network (RNN) refers to the *class* of models instead of the specific architecture of Elman (Eq. 1.13).

We denote a vector with bold lower case $\boldsymbol{x}$, and a component of it with x_i. Equivalently, $(x_i)_{i=1}^p$ also stands for a vector $\boldsymbol{x}$ of size p. A matrix is represented with bold upper case

$\boldsymbol{X}$, while a tensor of more than 2 modes is represented with a calligraphic bold capital $\boldsymbol{\mathcal{X}}$. $\boldsymbol{x}_i$ stands for the i-th row in matrix $\boldsymbol{X}$. $\boldsymbol{X} \in \mathbb{R}^{M \times N}$ denotes a two-mode matrix $\boldsymbol{X}$ as well as $\boldsymbol{X} \in \mathbb{R}^{(m_1 \cdot m_2) \times (n_1 \cdot n_2)}$ does. To avoid too many subscripts, we use round brackets in case multiple indices are needed, such as $\boldsymbol{X}(i,j)$ and $\boldsymbol{\mathcal{X}}(i,\bullet,\bullet)$. Denoting an identity matrix of size $N \times N$ as $\boldsymbol{E}^N$, the i-th row, $\boldsymbol{e}_i^N$, yields then a one-hot vector. Specifically in the context of SVD, we use a single integer r to refer to the first r columns of each factor matrix of $\boldsymbol{U}_r$ and $\boldsymbol{V}_r$. A superscript in rectangle brackets as in $\boldsymbol{x}_i^{[t]}$ denotes a time-stamped vector. We use square brackets $[\bullet,\bullet,...]$ to denote the concatenation operation; and $\times_n$ stands for the n-mode product as defined in [58]. The parameter set of a function is denoted as its subscript as $f_\Theta(\cdot)$.

1.1.3 A Framework for Representation Learning

Definition 1. *A predictive machine learning model that is based on latent representation can be decomposed into two sub-models: a* representation *model* $\xi(\cdot)$ *with a parameter set* Φ*, and a* prediction *model* $\eta(\cdot)$ *with a parameter set* Θ*:*

$$\hat{\boldsymbol{y}} = \eta_\Theta(\boldsymbol{z}), \text{ with } \boldsymbol{z} = \xi_\Phi(\boldsymbol{x}). \tag{1.1}$$

$\boldsymbol{x}$ denotes the raw feature input, which could take any form such as a vector, a matrix, a tensor, or even a sequence of them. We use the notation of a vector for the sake of simplicity, and also because the latter two formats can always be transformed into a vector. The representation model $\xi_\Phi(\cdot)$ transforms each raw input into a new latent vector $\boldsymbol{z}$. This latent representation of the raw input is expected to facilitate the parameterized prediction model $\eta_\Theta(\cdot)$ consuming $\boldsymbol{z}$ [11].

In earlier attempts in handling complex data features, the function $\xi_\Phi(\cdot)$ could be hand-designed, exploiting domain specific knowledge of human [8, 62]. These features are extracted from the raw data as a pre-processing step, and are typically denser in information while lower in dimensionality. Once extracted, they remain constant during the training of the prediction model $\eta_\Theta(\cdot)$. These procedures could be time-consuming and fail to scale to larger and novel data situations. More importantly, these hand-designed features are strictly limited by human knowledge.

There are also more generic classical approaches to generate representations. The well-known SVD, for instance, can be utilized to extract a smaller number r of orthogonal factors that are linear combinations of the raw data features: $\boldsymbol{X} \approx \boldsymbol{U}_r \boldsymbol{D}_r \boldsymbol{V}_r^T \Rightarrow \boldsymbol{U}_r =$

$\boldsymbol{X}\boldsymbol{V}_r\boldsymbol{D}_r$, where each row $\boldsymbol{U}_r(i)$ can be interpreted as the latent representation of raw data instance $\boldsymbol{X}(i)$. Applying the framework in Def. 1, it is easy to see that the latent vector $\boldsymbol{z}$ corresponds to $\boldsymbol{U}(i) = \xi_{\{\boldsymbol{V}_r,\boldsymbol{D}_r\}}(\boldsymbol{X}(i))$. That is to say, the representation model ξ_Θ is a linear function with parameters $\boldsymbol{V}_r$ and $\boldsymbol{D}_r$. In case that $\eta_\Theta(\cdot)$ is a linear regression, this whole setting becomes the Principal Component Regression (PCR) [52, 50]. To this end, PCR can be seen as a simple and intuitive form of enhancing predictive modeling using representation learning. However, the principal components can also serve as input to a subsequent logistic regression [64] or other prediction models.

Another more recent approach to extract generic latent representation is the autoencoder [110], whose simplest form is a two-layered feed-forward neural network: $\hat{\boldsymbol{x}} = \boldsymbol{W}\boldsymbol{z} + \boldsymbol{c}$, with $\boldsymbol{z} = \rho(\boldsymbol{V}\boldsymbol{x} + \boldsymbol{b})$ with the target being identical to the input. The model learns to first represent each input $\boldsymbol{x}$ as $\boldsymbol{z}$, and then attempts to reconstruct $\boldsymbol{x}$ as $\hat{\boldsymbol{x}}$ from $\boldsymbol{z}$. The function $\xi_\Phi(\cdot)$ is instantiated by the network with parameters $\{\boldsymbol{V}, \boldsymbol{b}\}$.

The representations generated by, e.g., an SVD and an autoencoder are identified by exploring the inner characteristics of the data features. One attempts to represent the complete features using only a few relevant factors while neglecting the less relevant ones. Therefore such methods are often also referred to as dimension reduction. These representations may remain constant, or they can function as initialization of the parameters and be adjusted while training $\eta_\Theta(\cdot)$ [66, 110].

Recent developments in machine learning, however, encourage the training of both the representation and prediction models jointly. In many cases, the pre-training of the representation model is either unnecessary or impossible. That is to say, both sets of parameters Θ and Φ in Def. 1 are optimized jointly w.r.t. one cost function:

$$\arg\min_{\Theta,\Phi} cost(\boldsymbol{y}, \hat{\boldsymbol{y}} \mid \Theta, \Phi) \text{ with } \hat{\boldsymbol{y}} = \eta_\Theta(\xi_\Phi(\boldsymbol{x})). \tag{1.2}$$

The representation function $\boldsymbol{\xi}_\Phi(\cdot)$ is enforced to be responsive to the predictive task. SVD and autoencoder, on the contrary, explores the inner structure of input feature $\boldsymbol{x}$, without any information regarding the prediction task.

Neural networks, especially those of deeper architectures, can be quite efficient for generating latent and dynamic representation. First, a feed-forward neural network, being a universal approximator, could theoretically mimic any continuous mapping functions [31, 48, 30]. Second, deep neural networks prove to generate latent representations of increasing abstraction levels [11]. The prediction model is therefore only exposed to the last and most abstract representation.

Furthermore, depending on various forms of the raw data, one could design more specific neural network architectures and they outperform many models based on hand-designed features. Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) are two of the most successful examples of this class of representation models.

A CNN [68] is based on the convolution operation. It assumes a spatially local correlation in the input feature space, and defines a corresponding connectivity pattern of the neurons. By re-using the same set of the locally connected weights convolving across the input features, CNNs can efficiently handle input feature with less weights in comparison with fully-connected networks. The most successful use cases are found in image classification, using multiple convolution layers with pooling and sub-sampling [61, 103].

RNNs, on the other hand, specialize in processing sequential data. One of the most popular variants is the Elman architecture [36], which has inspired many developments that followed. The core idea is to re-use the same weight matrix $\boldsymbol{W}$ to map the feature vector $\boldsymbol{x}^{[t]}$ at each time step to a hidden state $\boldsymbol{z}^{[t]}$. Each hidden state is then augmented with a mapping from the hidden state of the proceeding time step: $\boldsymbol{z}^{[t]} = \rho(\boldsymbol{W}\boldsymbol{x}^{[t]} + \boldsymbol{U}\boldsymbol{z}^{[t-1]} + \boldsymbol{b})$. A predictive model often consumes $\boldsymbol{z}^{[t]}$ as input, which is expected to represent the entire history of $\{\boldsymbol{x}^{[\leq t]}\}$ by the recurrent definition. However, due to the exploding and vanishing gradient problem [46, 12], this recurrent architecture may fail to learn long-term dependencies by error back-propagation from data. Long Short-Term Memory (LSTM) network [47, 40] provides a solution by introducing gating mechanisms into the calculation of the hidden state. A more lightweight version of LSTM, the Gated Recurrent Units (GRU), have proven to be equally powerful in storing input pattern over a long period of time. These RNNs nowadays provide the building blocks for sequence modeling tasks such as language modeling [54, 74], machine translation [6, 102] and reinforcement learning [113, 7].

The training of these deep architectures of neural networks with back propagation could be challenging, and would not have been feasible, if a variety of advanced training techniques hadn't been developed as well. They include regularizations [44, 98], optimizations [55, 43] and activations [79, 56].

There are a few known limitations of representation learning with neural networks [62]. First, the fact that the representation model responds to the predictive task also implies that these latent representations are task specific. Multi-task and transfer learning attempts to study this issue by defining multiple predictive models that share the same latent representation as input [19]. Second, in comparison to the hand-designed features,

the representation generated by neural networks are often not directly interpretable. Only in rare cases could one assign semantic interpretation to the single components in the latent representation [3, 118, 88].

1.2 Representation Learning in Knowledge Graphs

In this section, we first give a brief introduction to the special data situation in knowledge graph modeling. We further present a few examples of important relational learning models based on tensor decomposition, and demonstrate their connections to representation learning. We introduce a novel representation learning technique, which enables decomposition models to derive latent representations for *new* entities without being retrained.

1.2.1 Introduction

A Knowledge Graph (KG) is a graph structured data base that stores known facts about relations between entities [82]. The RDF standard [67] represents a fact as a triple of three types of entities: <`subject, predicate, object`>. For instance, the fact that George Lucas is the director of Star Wars IV could be stored as a triple <`George_Lucas, isDirectorOf, StarWars_IV`>. This standard is widely applied by a number of large scale KGs, such as Yago [100] and DBpedia [5]. Since these KGs are often incomplete, one of the most studied tasks is to predict the probability that an unknown fact involving known entities is true. This task is also referred to as link prediction or KG completion [82].

In Fig. 1.1 is an exemplary KG. There are three subjects: `George_Lucas`, `Star_Wars_IV` and `Raiders_Of_The_Lost_Ark` and the last two are also observed to be objects. Three probable types of relations are `isDirectorOf`, `isWriterOf` and `isOfGenre`. The corresponding triple store consists of 7 facts in total as listed in Fig. 1.1. The KG is demonstrated in three different formats: as a graph, as a tensor and as a triple store.

In statistical relational learning [62], one stores a collective of known fact triples as a three-way adjacency tensor such as in Fig. 1.1. Each entry in the tensor is assigned the value 1 if the fact it represents is known to be true. Otherwise the entry has value 0. A special case of such a tensor is a user-item matrix, where there is only one type of predicate, i.e., purchasing.

Decomposing such a three-way tensor yields a lower dimensional representation vector for each entity. These representation vectors are often expected to reveal the underlying

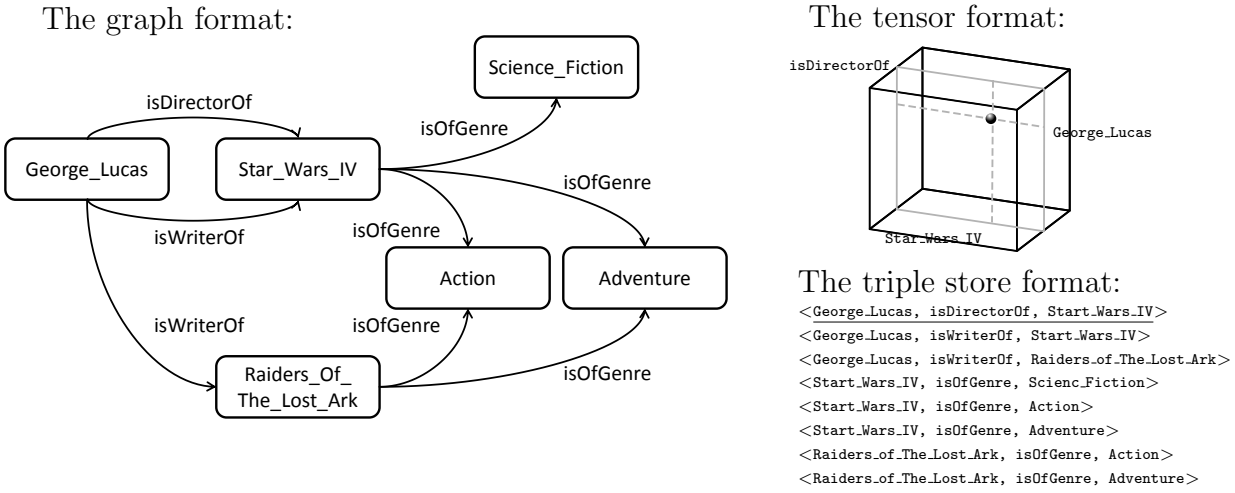


Figure 1.1: A Knowledge Graph example three formats: Left: as directed graph. Bottom right: as triple store. Top right: as binary tensor, where we only annotate the first triple for illustrative purposes.

characteristics of the entities in this latent space, e.g. the similarity between entities. If one reconstructs the tensor using the derived latent representations, the value at a position that indexes an unknown fact can be interpreted as the probability or confidence that this fact is true. In Appendix A.1 we provide a numerical example to illustrate the decomposition and reconstruction, i.e., the prediction of a user-item matrix.

Common choices of decomposition models are, for instance, HOSVD [58], CP/PARAFAC, Tucker [107], RESCAL [83] and multiway neural networks [34]. In the following, we review a number of the most representative decomposition models and, more importantly, reveal their relationship to representation learning.

1.2.2 Relational Learning Based on Tensor Decomposition

Now we provide a formal definition of probabilistic relational learning applying tensor decomposition. For the sake of generality, we consider relations consisting of three entities, including a subject, a predicate and an object. Consequently, it requires a database in three-way adjacency tensor form.

Definition 2 (Relational Learning with Tensor Decomposition)**.** *Given I subjects, J predicates and K objects, the target tensor is thus of shape $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I \times J \times K}$. One decomposes $\boldsymbol{\mathcal{Y}}$*

as

$$\begin{aligned}&\mathbb{P}(\boldsymbol{\mathcal{Y}}(i,j,k)=1)\sim\mathcal{D}(\pi_{i,j,k})\\&\quad\textit{with }\pi_{i,j,k}=f_\Theta(\boldsymbol{a}_i,\boldsymbol{b}_j,\boldsymbol{c}_k)\ \forall i\in[1,I],j\in[1,J],k\in[1,K]\\&\quad\textit{and }\boldsymbol{A}\in\mathbb{R}^{I\times r_A},\boldsymbol{B}\in\mathbb{R}^{J\times r_B},\boldsymbol{C}\in\mathbb{R}^{K\times r_C}.\end{aligned}\tag{1.3}$$

The distribution assumption $\mathcal{D}$ could be, e.g., Gaussian distribution in case of CP and Tucker, or Bernoulli distribution in case of multiway neural networks. The hyper-parameters r_A, r_B and r_C are often termed as the ranks of the decomposition. They define the dimensions of the latent vector representing each entity, and thus the complexity of the model. In case of, e.g., CP, Tucker and RESCAL, all three ranks are restricted to be the same; which may be relaxed for multi-way neural network.

Under a proper distribution assumption, the probabilistic tensor decomposition model learns two things from the data. First, it learns a representation vector for each of the subjects, predicates and objects, which form the matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$, respectively. Second, it learns a parameter set Θ in the function $f(\cdot,\cdot,\cdot)$ that joins the representation vectors to reconstruct the target tensor. Different decomposition models vary in their definition of the function $f(\cdot,\cdot,\cdot)$:

The CP/PARAFAC [53] decomposes of a three-mode tensor as:

$$f(\boldsymbol{a}_i,\boldsymbol{b}_j,\boldsymbol{c}_k)=(\boldsymbol{a}_i\circ\boldsymbol{b}_j)^T\boldsymbol{c}_k\tag{1.4}$$

where the parameter set of function f is empty: $\Theta=\emptyset$.

In case of decomposing $\boldsymbol{\mathcal{Y}}$ using Tucker [107], the function $f(\cdot,\cdot,\cdot)$ defines joining $\boldsymbol{a}_i$, $\boldsymbol{b}_j$ and $\boldsymbol{c}_k$ with the core tensor $\Theta=\boldsymbol{\mathcal{G}}$ as parameter

$$f_{\boldsymbol{\mathcal{G}}}(\boldsymbol{a}_i,\boldsymbol{b}_j,\boldsymbol{c}_k)=\boldsymbol{\mathcal{G}}\times_1\boldsymbol{a}_i^T\times_2\boldsymbol{b}_j^T\times_3\boldsymbol{c}_k^T.\tag{1.5}$$

RESCAL [83] is a special case of two-way Tucker [20]. It enforces an entity to be represented by one representation vector, regardless of whether it functions as subject or object in a relation triple. In Fig 1.1, for instance, the entity `Star_Wars_IV` is observed to be an object with predicate `isDirectorOf`, as well as a subject with predicate `isOfGenre`. In order to apply this constraint, one could query the representation vector for subject and object from the same matrix $\boldsymbol{A}$:

$$f_{\boldsymbol{\mathcal{G}}}(\boldsymbol{a}_{i_S},\boldsymbol{b}_j,\boldsymbol{a}_{i_O})=\boldsymbol{\mathcal{G}}\times_1\boldsymbol{a}_{i_S}^T\times_2\boldsymbol{b}_j^T\times_3\boldsymbol{a}_{i_O}^T,\tag{1.6}$$

where we use i_S and i_O to denote the indices of the subject and object, respectively.

A multi-way neural network concatenates the latent representations to serve as input to a neural network with one hidden layer.

$$f_{\{\boldsymbol{w},\boldsymbol{\mathcal{V}}\}}(\boldsymbol{a}_i, \boldsymbol{b}_j, \boldsymbol{c}_k) = \sigma(\boldsymbol{w}^T \sigma(\boldsymbol{V}[\boldsymbol{a}_i, \boldsymbol{b}_j, \boldsymbol{c}_k])). \tag{1.7}$$

In contrast to the previous linear models, a multi-way neural networks consists of non-linearity and is often trained with back propagation.

For large KGs, the tensor could be extremely large and sparse. It is therefore computational costly to train the decomposition of the entire tensor, including both known and unknown facts. However, training only on the known triples could easily lead to over-fitting [81]. In this case, the model would typically predict positive values for all probable triples. As a solution, one could apply parameter regularization to the decomposition model [59]. This is closely related to the concept of low-rank reconstruction of, e.g., SVD and CP, where one removes the less relevant factor columns to prevent a perfect reconstruction, i.e., over-fitting. Alternatively, one could sample a small proportion of unknown triples and include them into the training set [63], under the assumption that they are false facts.

1.2.3 Relational Learning and Representation Learning

Theorem 1. *A relational learning model based on tensor decomposition in Def 2 is an instance of the generic model framework in Def. 1. Specifically, the latent representation vectors $\boldsymbol{a}_i$, $\boldsymbol{b}_j$ and $\boldsymbol{c}_k$ are outputs of representation models defined by:*

$$\begin{aligned} \boldsymbol{a}_i &= g_{\boldsymbol{A}}(\boldsymbol{e}_i^I) = \boldsymbol{A}^T \boldsymbol{e}_i^I, \\ \boldsymbol{b}_i &= g_{\boldsymbol{B}}(\boldsymbol{e}_j^J) = \boldsymbol{B}^T \boldsymbol{e}_j^J, \\ \boldsymbol{c}_i &= g_{\boldsymbol{C}}(\boldsymbol{e}_k^K) = \boldsymbol{C}^T \boldsymbol{e}_k^K, \end{aligned} \tag{1.8}$$

where the matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ are identical with those in Def. 2

Taking the subject in Eq. 1.8 for example, $\boldsymbol{e}_i^I \in \{0,1\}^I$, being the i-th row in an identity matrix $\boldsymbol{E}^I$, is a one-hot vector. One one hand, the function $g_{\boldsymbol{A}}$ defines a simple feed-forward layer, whose parameter $\boldsymbol{A} \in \mathbb{R}^{I \times r_A}$ is often interpreted as a look-up table. By multiplying with $\boldsymbol{e}_i^I$, only the i-th row in the matrix, namely $\boldsymbol{a}_i$ is selected. Here the function $g_{\boldsymbol{A}}(\cdot)$ is an instance of $\xi(\cdot)$ defined in Eq. 1.1. The one-hot vectors $\boldsymbol{e}_i^I, \boldsymbol{e}_j^J, \boldsymbol{e}_k^K$ are raw input features that index the entities. On the other hand, the prediction model defined by function $f(\cdot,\cdot,\cdot)$ instantiates $\eta(\cdot)$ in Eq. 1.1. During training, the representation model

defined by $g(\cdot)$ and the prediction model by $f(\cdot,\cdot,\cdot)$ are adapted jointly to reconstruct the tensor.

The one-hot vector only indicates the entity of interest and selects the corresponding row in the look-up matrix. This function is sufficient –and in fact efficient– for a simple decomposition model. It does not, however, relates the entity to its features. There are two possible sources of entity features that can be included into the relational modeling. First, in some cases, additional features may be available in the data, such as personal information of the users or production information on the items. These entity features may serve as the input to the mapping function $g(\cdot)$ [39]. Second, even in absence of such additional information, the feature available on each entity is in fact the collective of facts that describe its known relations to other entities. In the following section we elaborate the second case and propose to learn representations from these known facts.

1.2.4 Representation Learning from Known Facts

The slices of $\boldsymbol{\mathcal{Y}}(i,\bullet,\bullet)$, $\boldsymbol{\mathcal{Y}}(\bullet,j,\bullet)$, and $\boldsymbol{\mathcal{Y}}(\bullet,\bullet,k)$, namely the i-th, j-th and k-th slices from the tensor, consist of all known facts about entity i as subject, j as predicate and k as object, respectively. Therefore, we propose to extract latent representations from these entity features.

Definition 3 (Representation Mapping)**.**

$$\begin{aligned}
\boldsymbol{a}_i &= g_{\boldsymbol{M}_A}(vec(\boldsymbol{\mathcal{Y}}(i,\bullet,\bullet))) = \boldsymbol{M}_A vec(\boldsymbol{\mathcal{Y}}(i,\bullet,\bullet)),\ \textit{with } \boldsymbol{M}_A \in \mathbb{R}^{r_A \times (J\cdot K)} \\
\boldsymbol{b}_j &= g_{\boldsymbol{M}_B}(vec(\boldsymbol{\mathcal{Y}}(\bullet,j,\bullet))) = \boldsymbol{M}_B vec(\boldsymbol{\mathcal{Y}}(\bullet,j,\bullet)),\ \textit{with } \boldsymbol{M}_B \in \mathbb{R}^{r_B \times (I\cdot K)} \\
\boldsymbol{c}_k &= g_{\boldsymbol{M}_C}(vec(\boldsymbol{\mathcal{Y}}(\bullet,\bullet,k))) = \boldsymbol{M}_C vec(\boldsymbol{\mathcal{Y}}(\bullet,\bullet,k)),\ \textit{with } \boldsymbol{M}_C \in \mathbb{R}^{r_C \times (I\cdot J)}.
\end{aligned} \tag{1.9}$$

As the notation suggests, we apply the same $g(\cdot)$ functions in Eq. 1.8, which again defines a feed-forward layer but with different weights. And instead of one-hot vectors, it now takes as inputs the vectorized slices $\boldsymbol{\mathcal{Y}}(i,\bullet,\bullet)$, $\boldsymbol{\mathcal{Y}}(\bullet,j,\bullet)$ and $\boldsymbol{\mathcal{Y}}(\bullet,\bullet,k)$. The latent representations, $\boldsymbol{a}_i$, $\boldsymbol{b}_j$ and $\boldsymbol{c}_k$ are joined with the same function $f(\cdot,\cdot,\cdot)$ as in Def. 2. In Fig. 1.2 we compare the standard decomposition model with one which learns the latent representation vectors from known facts. The major difference is that we replace the one-hot vector with the tensor slice that stores all known facts about the entity's relation to other entities.

The most obvious advantage of this new representation model is that, if one observes a new entity with some known relations to existing entities in the database, one can instantly

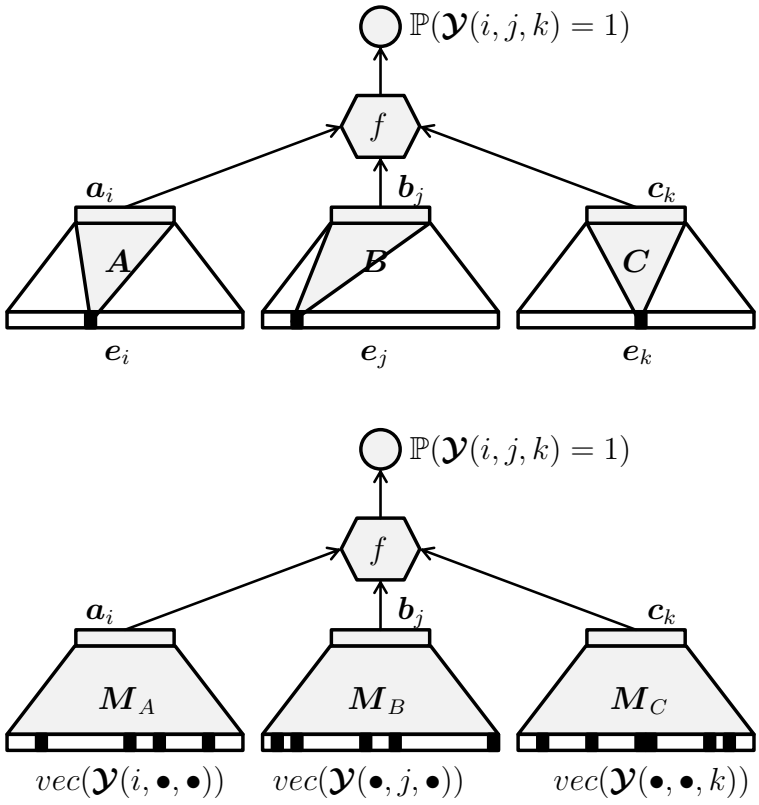


Figure 1.2: A comparison between two representation models in the context of tensor decomposition task. Top: corresponding to Eq. 1.8, each representation vector is generated as one row selected from the respective look-up matrices by a one-hot vector. Bottom: corresponding to Eq. 1.9, it takes as input a vectorized slice from the tensor itself.

derive the latent representation of this entity. Imagine a web shopping portal and a new customer who has just registered and purchased a few items. One would like to derive the customer's latent representation as soon as possible, in order to predict which other items the customer might be interested in. But in order to derive this single latent representation, one would need to add the new relational information into the database and inevitably retrain the decomposition model. However, such a single user with a few items is not likely to update the representation of other users or those of the items significantly, while retraining the decomposition could often be computational costly.

Our representation model in Def. 3 provides an efficient way to derive the latent representation of any new entity. The functions $g_{\boldsymbol{M}_A}(\cdot)$, $g_{\boldsymbol{M}_B}(\cdot)$ and $g_{\boldsymbol{M}_C}(\cdot)$ directly map from known facts to latent representations, giving us a good approximation of the latent

representation that *would have been* learned if one would retrain the model.

The input to function $g(\cdot)$ might in fact also include other entity attributes. Taking shopping portal as example, these might include age, gender, location etc. of the registered user. Also item attributes can be exploited to enable immediate recommendations to the potential users. This approach provides a solution to the well-known cold-start problem in recommendation system [39].

There are two technical challenges in learning mapping from known facts. Taking subject for example, the size of the input vector is $J \cdot K$, namely the product of the number of predicates and objectives, which could be huge for large knowledge graphs. Furthermore, the number of known relations involving subject i could be relatively small, making $\boldsymbol{\mathcal{Y}}(i, \bullet, \bullet)$ very sparse as well.

In section A.2 we proceed with the numerical example of user item matrix. We also prove this concept of representation mapping by introducing new entities.

1.2.4.1 Algorithms and Applications in Modeling Knowledge Graphs

Our published work, [114], proposes three different algorithms that learn to map from known facts to latent representation.

1. The first and most intuitive one is to fit a linear regression from the known facts to the learned latent representation *after* the tensor decomposition. This linear regression is independent of and thus compatible with any decomposition model.

2. We further proposed to integrate the regression fitting into the decomposition model in a every efficient way. We show that after each iteration of training the decomposition model, the representation vectors only need to be multiplied with a static matrix to satisfy the linear relation with the known facts. This algorithm is equivalent to optimizing two loss functions at the same time: one for the decomposition model and one for the linear regression.

3. As the third approach, we show that any decomposition model can be formulated as a feed-forward neural network. In other words, not only the multi-way neural networks, but also linear models such as CP and Tucker, can be trained with back propagation. This aspect greatly facilitates the representation mapping, because the linear regression becomes an additional linear layer. Such a model is end-to-end trainable with only one cost function. In this case, it is even unnecessary to store

the latent representations anymore, since the mapping is stored in form of the first layer of weight matrices.

We conduct experiments on three real-world data sets. In the first data set, we apply a decomposition of a user-item matrix of movies [42]. We simulate scenarios of new user and new items by masking different proportions known facts of these entities. Then we attempt to produce corresponding recommendations for both new users and new items. All three algorithms are exposed to these incomplete information, and map them into the latent space. Then we measure the quantity that the relational model can recover from the masked information. The 3rd algorithm turns out to outperform the most popular baseline model even with only half of the known facts.

Our second experiment has a similar setting, while using a subset of Freebase [14] data as a three-way tensor. We mask 20% of known facts and attempt to recover these with our model. As an upper bound we retrained the decomposition. The 3rd algorithm, the multiway neural network in combination with an additional linear layer, performs only 1.6% worse in AUROC and 9.1% worse in AUPRC compared with the upper bound. But note that in the latter case it usually takes a few seconds to perform the prediction, instead of multiple hours in case of retraining.

In the last experiment we focus on the quality and interpretability of the mapped representation. We compare the representations that are mapped from known facts to those that are learned with the recalculated decomposition. In the data set, we exploit the fact that the latent representations are promised to show strong column-wise correlation with a ground-truth matrix. We can confirm that, even when the representations are mapped instead of learned in the decomposition, their correlation with the ground-truth matrix is equally significant. Please note that in [114] we used the term *embedding* mapping instead of *representation* mapping.

1.3 Representation Learning in High Dimensional Sequential Data

In this section we study the second data situation that requires efficient representation learning: high dimensional sequential data. Sequential –or longitudinal– data are the outcomes of measuring the same features repeatedly at consecutive time steps. In electronic health record (EHR) data, for instance, patients with chronic diseases could be asked to regularly undergo a set of medical tests, which form a sequence of multidimensional feature vectors. Natural language data can also be handled as sequences. In each sentence, the feature space is a pre-defined vocabulary. The observed value at each time step is a single word represented by a one-hot vector, which is then mapped into a word representation (more often referred to as *embedding* in NLP literature) in the same fashion as Eq. 1.8. One major challenge with this data situation is that the sequences could be of variable lengths. In other words, the data instances have different input feature spaces, which makes it impossible to learn a direct mapping from the features to the target of modeling. Furthermore, the features observed at each time step could potentially also be of high dimensionality. Since they are repeated measurements on the same instance, applying conventional dimensional reduction techniques on each time step would violate the i.i.d. assumption. To this end, we also propose a novel architecture specialized in handling high dimensional sequential data.

1.3.1 Introduction

We denote multidimensional sequential data with an ordered set of time-stamped vectors as:

$$\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i}, \text{ with } \boldsymbol{x}_i \in \mathbb{R}^p. \tag{1.10}$$

At each time step t, one observes a feature vector $\boldsymbol{x}_i^{[t]}$ on instance i. The notation T_i indicates that the length of the sequence may vary for each data instance.

In order to perform predictive modeling based on these features, one requires techniques that extract fixed-size representation from the sequence. Simpler methods in representing these data may include information aggregation. A generic function to aggregate all time

steps up to time step T_i could be written as

$$\tilde{\boldsymbol{s}}_i^{[t]} = \sum_{t=1}^{T_i} \omega^{[t]} \boldsymbol{x}_i^{[t]} \in \mathbb{R}^p, \tag{1.11}$$

where ω is a weighting parameter. In case of $\omega^{[t]} = 1$ and $\frac{1}{T_i}$ the function defines a sum and average aggregation, respectively. Alternatively, $\omega^{[t]} = d(t, T_i)$ defines a weighting dependent on the distance d between each t and the current time step [119]. The bag-of-words solution in language modeling is a well-known instance of this model class. It represents a sentence or a paragraph by counting the frequency of each word in a vocabulary. One limitation of such an approach is that it disregards the order in which theses events appear. In language modeling, however, the order may encode the grammar information crucial in understanding the sentence.

[119, 37] apply this method also on clinical events data. Such an aggregation represents the number of multiple events that have been observed on each patient.

Another approach observes a fixed-size moving window of past Δ time steps as input features [9]:

$$\begin{aligned} \tilde{\boldsymbol{s}}_i^{[t]} &= [\boldsymbol{s}_i^{t-\Delta}, ..., \boldsymbol{s}_i^{t-2}, \boldsymbol{s}_i^{t-1}, \boldsymbol{s}_i^{t}], \\ \text{with} \quad \boldsymbol{s}_i^{[t]} &= g(\boldsymbol{x}_i^{[t]}) \ \forall t, \end{aligned} \tag{1.12}$$

where g again denotes a fully connected layer with probable non-linear activation. Compared to aggregation, the moving-window solution takes into account the order of events within the window by the concatenating operation. However, a small size of the window limits the model's capacity to capture long-term dependency. A large Δ would result in a very large representation $\tilde{\boldsymbol{s}}_i^{[t]}$. The representations generated from shorter sequences would thus contain large proportion of null information due to vacant inputs.

To conclude, an aggregation approach is invariant to the sequence length but neglects the order of events; while the moving window technique considers the order but is dependent on the sequence size in the window. Recurrent neural networks can join the advantages of both techniques, being capable of modeling long-term dependencies while taking into account the order in the sequence. They have gained much attention due to their success in modeling sequential data, from natural language data to reinforcement learning [41]. Also in medical domain, they have shown promising results and outperform more traditional methods in handling sequential data [38, 26].

1.3.2 Recurrent Neural Networks

One of the most popular RNN architectures is proposed by [36]:

$$\boldsymbol{h}_i^{[t]} = \rho(\boldsymbol{W}\boldsymbol{x}_i^{[t]} + \boldsymbol{U}\boldsymbol{h}_i^{[t-1]} + \boldsymbol{b}) \ \in \mathbb{R}^r. \tag{1.13}$$

In this setting, $\boldsymbol{h}_i^{[t]}$ denotes the so-called hidden state of the RNN, which consists of two additive terms. The first one is a mapping of the current input feature at t with a weight matrix $\boldsymbol{W}$, while the second term is a mapping from the last hidden state at $t-1$ using a weight matrix $\boldsymbol{U}$. $\rho(\cdot)$ denotes a non-linear activation function, such as tanh and sigmoid.

An RNN of this architecture could overcome the limitations of aggregation and moving window techniques, respectively. First, the hidden state is dependent on all previous inputs in a Markov fashion. The model is thus invariant to the number of the previous time steps. Second, by processing from $t = 0$ through $t-1$, the order in which these input vectors occur can be also taken into account by the representation $\boldsymbol{h}_i^{[t]}$ of a reasonable size. Another important aspect in RNNs is the fact that the same weight matrices $\boldsymbol{W}$ and $\boldsymbol{U}$ are reused at each time step. This enables the model to generalize to new time steps and to recognize the same input pattern when it re-occurs in the same sequence [11].

Though proven to be Turing complete with a finite size of the hidden state [94], such a simple architecture turns out to suffer from difficulties in learning long-term dependencies from data, due to the phenomenon of vanishing and exploding gradients [45, 46, 87].

To solve this problem, two major improvements on the Elman architecture have been proposed: Def. 4 describes the Long Short-Term Memory (LSTM) network, and Def. 5 the Gated Recurrent Unit. Both are now established state-of-the-art RNN models. They introduce gating mechanisms into the calculation of the hidden state, and have proven to be equally powerful in storing input pattern over a long period of time. [51], for instance, provides an empirical study in interpreting the gates in LSTM.

Definition 4 (Long Short-Term Memory [47, 40])**.**

$$\begin{aligned}
\boldsymbol{k}^{[t]} &= \sigma(\boldsymbol{W}^k\boldsymbol{x}^{[t]} + \boldsymbol{U}^k\boldsymbol{h}^{[t-1]} + \boldsymbol{b}^k) \\
\boldsymbol{f}^{[t]} &= \sigma(\boldsymbol{W}^f\boldsymbol{x}^{[t]} + \boldsymbol{U}^f\boldsymbol{h}^{[t-1]} + \boldsymbol{b}^f) \\
\boldsymbol{o}^{[t]} &= \sigma(\boldsymbol{W}^o\boldsymbol{x}^{[t]} + \boldsymbol{U}^o\boldsymbol{h}^{[t-1]} + \boldsymbol{b}^o) \\
\boldsymbol{g}^{[t]} &= \tanh(\boldsymbol{W}^g\boldsymbol{x}^{[t]} + \boldsymbol{U}^g\boldsymbol{h}^{[t-1]} + \boldsymbol{b}^g) \\
\boldsymbol{c}^{[t]} &= \boldsymbol{f}^{[t]} \circ \boldsymbol{c}^{[t-1]} + \boldsymbol{k}^{[t]} \circ \boldsymbol{g}^{[t]} \\
\boldsymbol{h}^{[t]} &= \boldsymbol{o}^{[t]} \circ \tanh(\boldsymbol{c}^{[t]}).
\end{aligned} \tag{1.14}$$

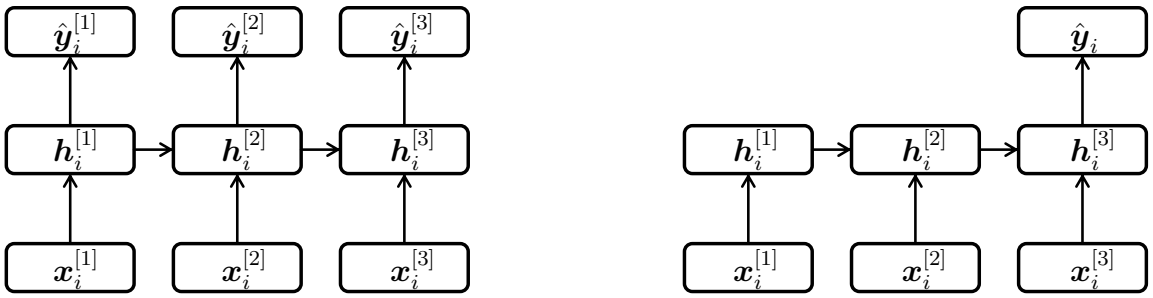


Figure 1.3: Illustration of two variants to utilize RNN as a representation model. In the left variant, the same prediction model is built on top of each time step's latent representation. In the right variant, only the last hidden state is exposed as input to the prediction model.

Definition 5 (Gated Recurrent Unit [27])**.**

$$\begin{aligned}
\boldsymbol{r}^{[t]} &= \sigma(\boldsymbol{W}^r \boldsymbol{x}^{[t]} + \boldsymbol{U}^r \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^r) \\
\boldsymbol{z}^{[t]} &= \sigma(\boldsymbol{W}^z \boldsymbol{x}^{[t]} + \boldsymbol{U}^z \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^z) \\
\boldsymbol{d}^{[t]} &= \tanh(\boldsymbol{W}^d \boldsymbol{x}^{[t]} + \boldsymbol{U}^d (\boldsymbol{r}^{[t]} \circ \boldsymbol{h}^{[t-1]})) \\
\boldsymbol{h}^{[t]} &= (1 - \boldsymbol{z}^{[t]}) \circ \boldsymbol{h}^{[t-1]} + \boldsymbol{z}^{[t]} \circ \boldsymbol{d}^{[t]}.
\end{aligned} \tag{1.15}$$

RNNs are applied as a representation model that generates as output the hidden states. It is expected that a hidden state is dependent on information of all previous time steps. There are two typical ways to deploy a prediction model on top of an RNN. One could build a predictive model, say $\hat{\boldsymbol{y}}_i^{[t]} = \rho(\boldsymbol{V}\boldsymbol{h}_i^{[t]})$ at each time step t, as is illustrated on the left side in Fig. 1.3. A well-known example of this setting is language modeling [75, 54], where one attempts to predict the probability of a next word given all previous ones. Alternatively, the predictive task could be performed based only on the last hidden state: $\hat{\boldsymbol{y}}_i = \rho(\boldsymbol{V}\boldsymbol{h}_i^{[T_i]})$. In this case, the last hidden state $\boldsymbol{h}_i^{[T_i]}$ is expected to have summarized information in the whole sequence, as is illustrated on the right side in Fig. 1.3. The predictive task could be, for instance, assigning a score or a class to the sequence. Examples are for instance a sentiment value to a sentence [4]; [60] performs classification tasks on sequences of 2D features that record pen coordinates, and sequences of spoken speech stored in 13 channels.

1.3.2.1 Application in Sequential EHR for Decision Support

In our published work, [116], we attempt to integrate patients' medical history as sequential features into a model that predicts individual therapy decisions. We extract thousands of *patient cases* defined with patient ID and a time step where a therapy decision was due. The challenging fact is that the patient cases all have a medical history of different lengths. They vary from 0 to 35 and are on average 4.1. At each time step we observe a feature vector of the shape 189, which consists of 6 types of clinical event features such as local recurrences, metastasis, therapies, etc. We use LSTM and GRU to encode these multidimensional sequences into one fixed-size latent representation. Following [38], we also include a representation vector learned from static patient features, such as demographic information and primary tumor, etc. Both representations are concatenated to serve as input to the decoder, which is a generalized version of multinomial hierarchical regression [108]. This hierarchical predictive model mimics the decision procedure of the physicians and generates predictions of therapy predictions. For comparison, we also implement a baseline representation model which is an aggregation function of the sequential features; and a baseline decoder which is a one-layered logistic regression. Our experiments demonstrate that the RNN model offers more significant contribution to modeling the decisions. Compared with the baseline, a combination of GRU with a logistic regression is already capable of doubling the prediction quality in term of AUPRC, while the hierarchical decoder further improves it by a smaller margin of 8.3%. This demonstrates that the RNN can extract from the medical history information that are more relevant for the therapy prescriptions.

A comparison between patients is also a useful feature in a decision support system. For each test patient, if one could provide a list of similar patients in the database, and show that these patients have received therapies similar to the prediction for this test patient, this would give the user, namely the physician, much confidence in interpreting the prediction. However, such a comparison is not trivial in the input space –since each patient has a different input space. We show that one can instead compare patients in the latent space learned by the RNN model. We use a k-NN classifier exposed to learned representations, and compare its predictions with those which are generated with the hierarchical and logistic regression. Training patient cases, which are identified as similar in the latent space by the k-NN classifier, turn out to have received similar therapies as our model predicts.

1.3.3 Tensor-Train Layer

The challenge that the sequential features could also be of high dimensionality has not yet been well addressed in recent works on RNNs. One probable reason is that most of the successful works apply RNNs to model natural language data through word embeddings [73, 76]. The size of the embeddings is a hyper-parameter free to tune and obviously, no one is tempted to choose a word embedding size that the RNN model cannot handle. However, in other data situations such as sensor and video data, the sequential features could be of much larger dimensionality that cannot be tuned.

Furthermore, we argue that high dimensionality is a concept relative to the information content. A feature vector that has a high sparsity, or contains a high proportion of redundant information, could also be seen as high dimensional. It leads to over-parameterization of the weights, i.e., a large proportion of weights cannot be sufficiently adjusted during the training.

For non-sequential features, many dimension reduction techniques are available, such as SVD and random projection. These methods cannot be applied directly on the feature vector at each different time step without violating the i.i.d. assumption. The state-of-the-art solution borrows the idea of word embedding. It learns a mapping from each feature vector $\boldsymbol{x}_i^{[t]}$ to a lower dimensional space similar to Eq. 1.12. But instead of concatenating the low dimensional representations, one feeds them to an RNN model. This approach is, for instance, successfully applied in [38, 26] to handle the high dimensional patient features. However, it introduces one more layer of weights that could potentially also be large. In contrast to word embedding layers, where only one indexing operation is necessary to query a row in the weight matrix, one would have to calculate the complete matrix multiplication.

To this end, we propose a novel solution to the high dimensionality and/or high sparsity and redundancy in sequential data, by integrating a Tensor-Train layer [85] into RNN architectures.

The Tensor-Train Decomposition

Tensor-Train is a tensor decomposition model being able to scale to arbitrary number of orders.

Definition 6 (Tensor-Train Decomposition [86]). *The target d-way tensor* $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{p_1 \times p_2 \times \ldots \times p_d}$

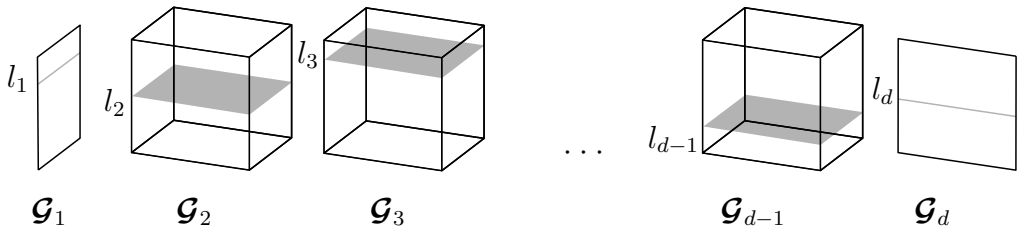


Figure 1.4: Illustration of Tensor-Train decomposition. One queries the l_k-th slice from the k-th core tensor $\boldsymbol{\mathcal{G}}_k$. The slices are then joined by inner product.

is to be decomposed as

$$\begin{aligned} \widehat{\boldsymbol{\mathcal{A}}}(l_1, l_2, ..., l_d) &= \boldsymbol{\mathcal{G}}_1(l_1) \cdot \boldsymbol{\mathcal{G}}_2(l_2) \cdot ... \cdot \boldsymbol{\mathcal{G}}_d(l_d) \\ \textit{where} \;\; \boldsymbol{\mathcal{G}}_k &\in \mathbb{R}^{p_k \times r_{k-1} \times r_k}, \;\; l_k \in [1, p_k] \;\; \forall k \in [1, d] \;\; \textit{and} \;\; r_0 = r_d = 1, \end{aligned} \tag{1.16}$$

namely into a set of so-called core tensors $\boldsymbol{\mathcal{G}}_k$ *for* $k = 1, 2, ..., d$.

Each entry $\boldsymbol{\mathcal{A}}(l_1, l_2, ..., l_d)$ is decomposed by a train of multiplication of the l_k-th slice from the core tensor $\boldsymbol{\mathcal{G}}_k$ (as is illustrated in Fig 1.4). The $(r_k)_{k=1}^d$ denotes the ranks of decomposition. The first and last ranks, r_0 and r_d, are restricted to be 1, so that the *train* of multiplications produces an scalar. In comparison with, e.g., CP and Tucker where each latent representation has only one rank, it has for $k = 1$ to $d - 1$ two ranks: r_{k-1} and r_k in Tensor-Train configuration. This allows for multiplication of joining a latent representations in a *train*. It easy to see that, as the tensor grows in its number of order d, one only needs to include more core tensors.

The Tensor-Train Layer

[85] proposes that the weight matrix in a fully connected layer can be reshaped into a high order tensor and then decomposed using Tensor-Train.

The usual way to denote the feed-forward pass based on the weight matrix $\boldsymbol{W}$ is formulated as

$$\begin{aligned} &\hat{\boldsymbol{y}}(j) = \sum_{i=1}^{M} \boldsymbol{W}(j, i) \cdot \boldsymbol{x}(i) + \boldsymbol{b}(j) \\ &\text{with } \boldsymbol{x} \in \mathbb{R}^M, \boldsymbol{y} \in \mathbb{R}^N, \forall j \in [1, N], \boldsymbol{W} \in \mathbb{R}^{N \times M}. \end{aligned} \tag{1.17}$$

The fact that the weights are often stored in a 2-way matrix $\boldsymbol{W}$ is to facilitate the matrix notation convenience of $\hat{\boldsymbol{y}} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$.

In contrast, we propose in Eq. 1.18 an equivalent but different way to write the layer, which is instead to facilitate the derivation of Tensor-Train layer:

$$\begin{aligned} &\hat{\boldsymbol{y}}(j) = \sum_{i=1}^{M} \boldsymbol{w}^*(l) \cdot \boldsymbol{x}(i) + \boldsymbol{b}(j) \\ &\text{with } \boldsymbol{w}^* = vec(\boldsymbol{W}) \in \mathbb{R}^{N \cdot M} \text{ with } l = j \cdot M + i. \end{aligned} \tag{1.18}$$

The weight matrix is reshaped into a vector of length $N \cdot M$. The reshaped weight vector, $\boldsymbol{w}^*$, consists of N blocks of vectors of length M. Since there exists a bijection between l and (j, i) as in Eq. 1.17, we can use l to query the weights. To calculate the j-th output, one queries the j-th block, and calculates the grand sum of its Hadamard product with input $\boldsymbol{x}$. The difference to Eq. 1.17 is that we query the corresponding weights as a block in a vector, instead of querying them as a row in a matrix. This formulation is illustrated in the upper graphic in Fig. 1.5.

We generalize this formulation now to two dimensional cases. When the input and output are 2 way matrices instead of 1 way vectors, namely $\boldsymbol{X} \in \mathbb{R}^{m_1 \times m_2}$, $\hat{\boldsymbol{Y}} \in \mathbb{R}^{n_1 \times n_2}$, the forward pass would correspondingly be

$$\begin{aligned} &\hat{\boldsymbol{Y}}(j_1, j_2) = \sum_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} \boldsymbol{W}^*(l_1, l_2) \cdot \boldsymbol{X}(i_1, i_2) \\ &\text{with } l_1 = j_1 \cdot m_1 + i_1, l_2 = j_2 \cdot m_2 + i_2, \boldsymbol{W}^* \in \mathbb{R}^{(m_1 \cdot n_1) \times (m_2 \times n_2)}, \end{aligned} \tag{1.19}$$

which is illustrated in the lower graphic in Fig. 1.5. The weights are thus stored in a 2 way matrix consisting of $n_1 \times n_2$ 2D blocks of size $m_1 \times m_2$. Analogous to Eq. 1.18, in order to calculate the entry (j_1, j_2) in the output $\hat{\boldsymbol{Y}}$, one queries the (j_1, j_2)-th block in $\boldsymbol{W}^*$.

The input, output and the weights can be thus generalized into d-way tensors:

Theorem 2. *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m_1 \times m_2 \times \ldots \times m_d}$ and $\widehat{\boldsymbol{\mathcal{Y}}} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d}$ be the input and output tensors, respectively. And the weights are also stored in a d-way tensor of*

$$\boldsymbol{\mathcal{W}} \in \mathbb{R}^{(m_1 \cdot n_1) \times (m_2 \cdot n_2) \times \ldots \times (m_d \cdot n_d)}.$$

The forward pass takes the form of

$$\begin{aligned} &\widehat{\boldsymbol{\mathcal{Y}}}(j_1, j_2, ..., j_d) = \sum_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} ... \sum_{i_d=1}^{m_d} \boldsymbol{\mathcal{W}}(l_1, l_2, ..., l_d) \cdot \boldsymbol{\mathcal{X}}(i_1, i_2, ..., i_d) + \boldsymbol{\mathcal{B}}(j_1, j_2, ..., j_d) \\ &\textit{with } \; l_k = m_k \cdot j_k + i_k \; \forall k \in [1, d]. \end{aligned} \tag{1.20}$$

Note that in Theorem 2, as well as in Eq. 1.18 and Eq. 1.19, the index l_k is a function of indices j_k and i_k, which is omitted for a more concise notation.

Theorem 2 suggests that the mapping from a d-way tensor to another d-way tensor can be realized by a third d-way weight tensor. Even if the input and output are not d-way tensors, one could always reshape them to be such. The purpose is to decompose d-way weight tensor with Tensor-Train as defined in Eq. 1.16. This gives us the definition of Tensor-Train layer:

Definition 7 (Tensor-Train Layer [85])**.** *In a fully-connected layer $\hat{\boldsymbol{y}} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ with $\boldsymbol{x} \in \mathbb{R}^M$ and $\hat{\boldsymbol{y}} \in \mathbb{R}^N$, if N and M can be factorized into two arrays of the same length, namely $N = n_1 \times n_2 \times ... \times n_d$ and $M = m_1 \times m_2 \times ... \times m_d$, they can be reshaped into two d-way tensors respectively. The d-way tensor of weight can be decomposed using Tensor-Train, namely*

$$\widehat{\boldsymbol{\mathcal{Y}}}(j_1, j_2, ..., j_d) = \sum_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} ... \sum_{i_d=1}^{m_d} \boldsymbol{\mathcal{G}}_1(l_1) \cdot \boldsymbol{\mathcal{G}}_2(l_2) \cdot ... \cdot \boldsymbol{\mathcal{G}}_d(l_d) \cdot \boldsymbol{\mathcal{X}}(i_1, i_2, ..., i_d) + \boldsymbol{\mathcal{B}}(j_1, j_2, ..., j_d), \tag{1.21}$$

We denote such a layer with $\hat{\boldsymbol{y}} = TTL(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{b})$, or $\hat{\boldsymbol{y}} = TTL(\boldsymbol{x}|\boldsymbol{W})$ in absence of the bias.

The calculation of a Tensor-Train layer can be implemented as a computation graph in frameworks such as in Theano [104] and TensorFlow [1]. In Appendix B, we provide pseudo-codes of the forward pass algorithm. The Tensor-Train layer can be trained using back propagation, which requires the gradient information w.r.t. each slice in each core tensor.

Theorem 3. *The Jacobian matrix w.r.t. a slice in a core tensor, i.e., $\boldsymbol{\mathcal{G}}_k(l_k) \in \mathbb{R}^{r_{k-1} \times r_k}$ in a Tensor-Train layer is:*

$$\frac{\partial \hat{\boldsymbol{\mathcal{Y}}}(j_1, j_2, ..., j_d)}{\partial \boldsymbol{\mathcal{G}}_k(l_k)} = \sum_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} ... \sum_{i_d=1}^{m_d} \prod_{k^*>k} \boldsymbol{\mathcal{G}}_{k^*}(l_{k^*}) \cdot \boldsymbol{\mathcal{X}}(i_1, i_2, ..., i_d) \cdot \prod_{k^*<k} \boldsymbol{\mathcal{G}}_{k^*}(l_{k^*}) \tag{1.22}$$

In Appendix C we also provide the proof and an illustrative numerical example.

The most essential idea of Tensor-Train layer is, therefore, to reconstruct the full weight matrix, which is expected to be large, using a relatively much smaller number of parameters, namely the core tensors $\{\boldsymbol{\mathcal{G}}_k\}_{k=1}^d$. The Tensor-Train decomposition of the corresponding d-way weight tensor requires only $\sum_{k=1}^d m_k n_k r_{k-1} r_k$ parameters instead of $\prod_{k=1}^d m_k n_k$. It is

recommended to factorize the number of features M into a sequence of numbers of similar orders of magnitude. Should it not be the case or, should M be a prime number, one could add a few zero columns to the input feature vector.

The tensorization and decomposition of a layer enables it to efficiently consume high dimensional input features. However, caution has to be taken that the approximate reconstruction of the full weight matrix poses certain constraint on the expressiveness of the layer. Such constraint can be tuned by the rank parameters $\{r_k\}_{k=0}^{d}$. However, a too large rank can of course lead to larger core tensors and therefore longer training time. Therefore, the tensorization of the large weight matrices is especially effective, if one expects sparsity and redundancy in the input. This is the case with, e.g. feature maps generated by convolutional layers and raw pixel values [85], as well as binary coding of categorical features [115]. In these cases, the Tensor-Train layer can perform high dimensional mappings with fewer parameters. On the other hand, restricting the expressiveness may also realize some kind of regularization. Like CP and Tucker, Tensor-Train is also a low-rank decomposition of the target tensor. We therefore speculate that, in reconstructing the full weight matrix with low-rank core tensors, one could –to some extent– remove the over-parameterization effect.

1.3.4 Embedding Tensor-Train Layer into RNNs

It is straightforward to decompose the weight matrix mapping from input to hidden state in Elman RNN with Tensor-Train. In case of LSTM, there are four matrices $\boldsymbol{W}^k$, $\boldsymbol{W}^f$, $\boldsymbol{W}^g$ and $\boldsymbol{W}^o$ in Eq. 1.14 to be decomposed.

In the implementation of plain LSTM, it is often more efficient to calculate

$$\boldsymbol{W}\boldsymbol{x}^{[t]} = [\boldsymbol{W}^k, \boldsymbol{W}^f, \boldsymbol{W}^g, \boldsymbol{W}^o]\boldsymbol{x}^{[t]}, \tag{1.23}$$

namely the multiplication of input with the concatenation of these four matrices. By exploiting the parallel power of modern processing units, one could perform the four multiplications simultaneously, decreasing the run time by increasing the memory usage.

Therefore, we propose to decompose the concatenated matrix $\boldsymbol{W}$ instead of each of the four matrices. The simplest way is to multiply an arbitrary n_k in $(n_k)_{k=1}^{d}$ by the factor of 4. In case of GRU in Eq. 1.15, the factor is 3 because of the 3 gate weights of $\boldsymbol{W}^r$, $\boldsymbol{W}^z$ and $\boldsymbol{W}^d$. Applying Def 7 and the corresponding notation $TTL(\cdot|\cdot)$, we present Tensor-Train LSTM and Tensor-Train GRU as follows.

Definition 8 (Tensor-Train Long Short-Term Memory)**.**

$$\begin{aligned}
&\boldsymbol{k}^{[t]} = \sigma(\boldsymbol{v}^{k,[t]} + \boldsymbol{U}^k \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^k) \\
&\boldsymbol{f}^{[t]} = \sigma(\boldsymbol{v}^{f,[t]} + \boldsymbol{U}^f \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^f) \\
&\boldsymbol{o}^{[t]} = \sigma(\boldsymbol{v}^{o,[t]} + \boldsymbol{U}^o \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^o) \\
&\boldsymbol{g}^{[t]} = \tanh(\boldsymbol{v}^{g,[t]} + \boldsymbol{U}^g \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^g) \\
&\boldsymbol{c}^{[t]} = \boldsymbol{f}^{[t]} \circ \boldsymbol{c}^{[t-1]} + \boldsymbol{k}^{[t]} \circ \boldsymbol{g}^{[t]} \\
&\boldsymbol{h}^{[t]} = \boldsymbol{o}^{[t]} \circ \tanh(\boldsymbol{c}^{[t]}). \\
&\textit{with} \quad [\boldsymbol{v}^{k,[t]}, \boldsymbol{v}^{f,[t]}, \boldsymbol{v}^{o,[t]}, \boldsymbol{v}^{g,[t]}] = TTL(\boldsymbol{x}^{[t]} \mid [\boldsymbol{W}^k, \boldsymbol{W}^f, \boldsymbol{W}^g, \boldsymbol{W}^o]).
\end{aligned} \tag{1.24}$$

Definition 9 (Tensor-Train Gated Recurrent Unit)**.**

$$\begin{aligned}
&\boldsymbol{r}^{[t]} = \sigma(\boldsymbol{v}^{r,[t]} + \boldsymbol{U}^r \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^r) \\
&\boldsymbol{z}^{[t]} = \sigma(\boldsymbol{v}^{z,[t]} + \boldsymbol{U}^z \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^z) \\
&\boldsymbol{d}^{[t]} = \tanh(\boldsymbol{v}^{d,[t]} + \boldsymbol{U}^d (\boldsymbol{r}^{[t]} \circ \boldsymbol{h}^{[t-1]})) \\
&\boldsymbol{h}^{[t]} = (1 - \boldsymbol{z}^{[t]}) \circ \boldsymbol{h}^{[t-1]} + \boldsymbol{z}^{[t]} \circ \boldsymbol{d}^{[t]} \\
&\textit{with} \quad [\boldsymbol{v}^{r,[t]}, \boldsymbol{v}^{z,[t]}, \boldsymbol{v}^{d,[t]}] = TTL(\boldsymbol{x}^{[t]} \mid [\boldsymbol{W}^r, \boldsymbol{W}^z, \boldsymbol{W}^d]).
\end{aligned} \tag{1.25}$$

Applying Tensor-Train decomposition on the concatenated weight matrices further reduces the number of parameters, because it is easy to see that

$$\sum_{k=1}^{d} m_k n_k r_{k-1} r_k + (c-1)(m_1 n_1 r_0 r_1) < c \cdot \sum_{k=1}^{d} m_k n_k r_{k-1} r_k, \tag{1.26}$$

where c is the number of gates necessary. $c = 4$ in case of LSTM and $c = 3$ for GRU.

As a numerical example, assume an input of dimension $160 \times 120 \times 3$, which is a typical size of a video frame as in [117], and a hidden state of size 256. A fully-connected layer would contain 14,745,600 weights. If we apply Tensor-Train decomposition with factorizing the input size to be $8 \times 20 \times 20 \times 18$, the hidden state size to be $4 \times 4 \times 4 \times 4$ and a rank of 4, one would need 11,904 parameters in total for all 4 separate weight matrices in LSTM. But if the 4 weight matrices are concatenated and jointly decomposed, one only needs 2,976 parameters.

In contrast to a related and independent work [106], we assume that the hidden state is of much smaller dimensionality and therefore see no necessity in decomposing the transition matrices $\boldsymbol{U}$'s.

1.3.4.1 Application in Video Classification

In our published work, [117], we demonstrate the prowess of Tensor-Train RNNs in modeling video data. Video data can be seen as sequences of images, which are 3 way tensor of x-pixels, y-pixels and 3 RGB channels. Although neural networks, especially the deeper convolutional ones, have been showing promising performances in processing image data, a direct transfer of these models to video data remains a challenging task. Training deep CNN from scratch on each frame of image could result in extremely heavy computation. Applying pre-trained CNNs [93, 33] is prone to over-specification, because one could not always assume that the object patterns in a video to be similar with –if not the same as– those in the data used for pre-training. Furthermore, since the video is expected to also contain temporal patterns among images, one would need recurrent models to join the representation learned from each frame. This could be even more challenging, especially for longer sequences, to jointly train RNN and CNN at the same time.

We propose to apply Tensor-Train RNNs on video data mainly based on two considerations: First, [85] already demonstrates that Tensor-Train layers can consume efficiently RGB pixels in image classification tasks. Second, we exploit the fact that video data contain large amount of redundant pixels on each frame. We show that Tensor-Train RNNs can map the raw pixels into the hidden state using a much smaller number of core-tensors. These hidden states, i.e., the latent representation of each frame, are joined as in LSTM and GRU.

The experiments, conducted on three real-world benchmark video data sets, all develop the same conclusion: Compared with plain GRU and LSTM, decomposing the weight matrix mapping from input to hidden layer reduces the number of parameters by up to 4 orders of magnitude, while significantly improving the prediction accuracy. The probable reason is that the huge number of parameters are simply under-trained with the data samples. In comparison with state-of-the-art works, our implementations always demonstrate second highest accuracy scores in total, and the highest scores considering only neural network based models.

Tensor-Train RNNs provide an efficient building block whenever high dimensional sequences with potentially redundant information are expected. In field such as NLP, RNNs have been rather successful in handling the sequential aspect of the data and have inspired quite a number of promising architectures such as attention mechanism, stacked RNNs, deep RNNs, etc. Tensor-Train RNNs thus enable transferring all these successful architectures into new fields where RNNs have failed earlier, and open up a large number of

possibilities for future researches.

1.3.4.2 Application in Sequential EHR for Survival Prediction

In our published work, [115], we include survival analysis into the framework of representation learning model. First, a Tensor-Train RNN encodes the medical history of a patient case into a latent representation. Second, the prediction model is instantiated as an Accelerated Failure Time (AFT) model with Log-Normal distribution assumption in a similar fashion to generalized regression. This model architecture aims at predicting the Progression-Free-Survival (PFS) time in days, by taking into account the sequential and static features of each individual patient. In contrast to [116], we extract here different patient cases, namely situations where a progression –either local recurrence or metastasis– is observed at the next known time step. With Tensor-Train RNNs we focus on the challenge of sequential features that are not only high dimensional, but also sparse due to binary coding of categorical features.

As weak baselines, we apply traditional survival analysis models such as Cox-Regression and plain AFT model exposed to aggregated sequential and static features. These models produce sub-optimal predictions, possibly due to the fact that the aggregation of sequential features cannot represent the data in an effective way. Further weak baseline models are plain LSTM and GRU that directly consume the sparse sequences as in [116]. We also implement state-of-the-art solutions handling high-dimensional sequential features, by reducing the input size with a simple feed-forward layer [26, 38]. This solution functions as our strong baseline model.

We measure the prediction quality using Median Average Error on the original scale of the output, and the R^2 coefficient on the log scale. Our proposed model outperforms the strong baseline models by a margin of 2.9%, while requiring on average only 4.0% as many parameters as the strong baseline models. This factor is reduced to 1.2% in comparison with plain RNNs.

We show that the framework of representation learning of Def. 1 may combine the strengths of representation learning and the classical generalized regression models (GRM). Deep neural networks are proven to be promising in generating representation from less structured data features. The GRMs, on the other hand, are more careful and flexible in the distribution assumption of the target variable. We show that it is possible to expose GRM to the latent representation produced by deep neural networks, and train the entire model end-to-end.

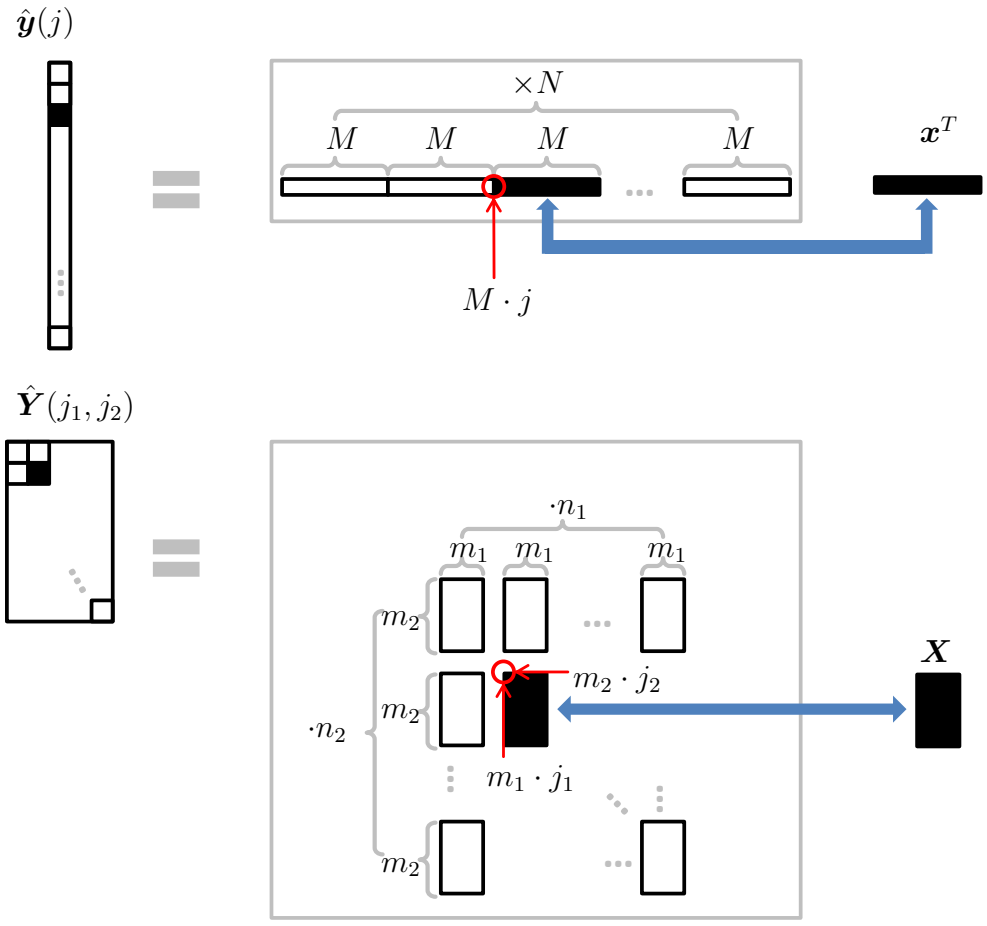


Figure 1.5: Special cases of Theorem 2 with $d = 1$ (upper) and $d = 2$ (lower). Assume a fully connected layer $\boldsymbol{y}(j) = \boldsymbol{W}(j)\boldsymbol{x}$. Upper: as Eq. 1.18 suggests, we reshape the weight matrix into a vector, and we need an index $l = M \cdot j + i$ to query the j-th block of corresponding weights vector to perform the forward pass. Lower: as Eq. 1.19, we reshape $\boldsymbol{x}$ and $\boldsymbol{y}$ into matrices, and we need indices of $(l_1 = m_1 \cdot j_1 + i_1, l_2 = m_2 \cdot j_2 + i_2)$ to query the block of weights –this time yielding a matrix. The blue arrow denotes the grand sum of the Hadamard product. This novel perspective of seeing a fully-connected layer forms the derivation of the Tensor-Train layer, where an arbitrary d might be chosen.

Chapter 2

Representation Mapping: Algorithms and Applications

Embedding Mapping Approaches for Tensor Factorization and Knowledge Graph Modelling

Yinchong Yang[2( )], Cristóbal Esteban[1,2], and Volker Tresp[1,2]

[1] Siemens AG, Corporate Technology, Munich, Germany
[2] Ludwig-Maximilians-Universität München, Munich, Germany
yinchong.yang@hotmail.com

Abstract. Latent embedding models are the basis of state-of-the art statistical solutions for modelling Knowledge Graphs and Recommender Systems. However, to be able to perform predictions for new entities and relation types, such models have to be retrained completely to derive the new latent embeddings. This could be a potential limitation when fast predictions for new entities and relation types are required. In this paper we propose approaches that can map new entities and new relation types into the existing latent embedding space without the need for retraining. Our proposed models are based on the observable —even incomplete— features of a new entity, e.g. a subset of observed links to other known entities. We show that these mapping approaches are efficient and are applicable to a wide variety of existing factorization models, including nonlinear models. We report performance results on multiple real-world datasets and evaluate the performances from different aspects.

1 Introduction

Latent embedding models, aka *factorization models*, have proven to be powerful approaches for modelling Knowledge Graphs (KG) as described in [17,18]. A special case is Collaborative Filtering (CF) where latent embedding models have shown state-of-the-art performance [16]. The common key aspect of these models is that an observed link between multiple entities can be modelled as the interaction between their latent embedding vectors. Multi-linear models such as CP/PARAFAC [14] and Tucker [22] as well as RESCAL [19] are typical examples of models that use latent embeddings. Nonlinear Neural Network-based embedding models are derived in [8,20]. For a more detailed review of these works please see [18].

The latent embedding vectors can be used in several ways. For example, it has been shown that distances between entities in the latent space are more compact and meaningful than in the original observable feature space. Also, in entity resolution, entities close to each other in the latent space can sometimes be interpreted as duplicates [7]. Finally, it has been shown that unknown links between known entities can be predicted based on interactions of their latent embeddings [18].

© Springer International Publishing Switzerland 2016
H. Sack et al. (Eds.): ESWC 2016, LNCS 9678, pp. 199–213, 2016.
DOI: 10.1007/978-3-319-34129-3_13

A drawback of latent embedding models is that they need to be retrained when new entities are appended to the database. For large-scale databases and/or situations where the system is expected to perform immediate operations, such as entity resolution or link prediction on the new entities, this would be very costly and factorization models would find only limited applications.

In this paper, we propose a new class of approaches to handle new entities and new relation types by mapping them into the latent space learned by the factorization model. We emphasize that such mapping models can be learned in conjunction with the training of the factorization model. To map a new entity into the latent space we only require the observable features of the entity. In a KG, for instance, such observable features form a binary vector or matrix, representing the existence of links between this a entity and a subset of known entities in the database.

The rest of the paper is organized as follows: In Sect. 2 we give a brief review of selected embedding-based factorization models and illustrate the concept of an embedding mapping. We show that for certain specific factorization models there exist embedding mappings in closed form. In Sect. 3, we propose a general framework that describes a variety of factorization models on a more abstract level and derive a framework defining the mapping models and elaborate three options for training. In Sect. 4 we present experimental results on real-world datasets. Section 5 discusses related work and Sect. 6 contains conclusions and an outlook for further works.

Notations: A matrix $\boldsymbol{A}$ is represented as a bold capital letter and a multidimensional tensor $\boldsymbol{\mathcal{X}}$ by a calligraphic bold capital letter. By default we assume a 3-dimensional tensor. In some applications the dimensions correspond to entities and relation types, which we sometimes treat as generalized entities. A matrix with indexing superscript as $\boldsymbol{A}^{(l)}$ denotes the latent embedding matrix for entities of the l-th dimension of a matrix or tensor. The matrix derived by unfolding a tensor w.r.t. dimension l is noted using subscripts as $\boldsymbol{X}_{(l)}$. Note that unfolding a matrix w.r.t. first and second dimension is equivalent to the matrix itself and its transpose, respectively. $\boldsymbol{X}^{\dagger}$ stands for the Moore-Penrose pseudoinverse. A vector is denoted with bold small letters such as $\boldsymbol{x}_{i,\bullet}$ and refers to the i-th row in a corresponding matrix $\boldsymbol{X}$. We refer to a set using either a simple capital Greek letter such as Θ or —if we focus on the elements— using curly brackets as $\{\boldsymbol{A}^{(l)}\}_{l=1}^{L}$. The concatenation operation is noted with squared brackets $[\bullet, ..., \bullet]_{+}$.

2 Factorization Models with Closed-Form Mappings

In this section we review a few well-studied factorization models that are based on latent embeddings and motivate our problem setting of mapping new entities into the latent embedding space.

Matrix Cases: First we review the Singular Value Decomposition (SVD) as a latent embedding model: For an SVD in form of $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{T}$ we interpret

the matrix $\boldsymbol{U}$ to consist of latent embedding vectors in rows, for each entity represented in the first dimension of $\boldsymbol{X}$. The matrix $\boldsymbol{X}$ is constructed by a linear combination of the embeddings $\boldsymbol{U}$ with weights defined as rows of $(\boldsymbol{DV}^T)^T$.

Then we consider $\boldsymbol{U} = \boldsymbol{X}(\boldsymbol{DV}^T)^\dagger$, which is an inverse-relation, to be a mapping function from $\boldsymbol{X}$ to the latent embedding in $\boldsymbol{U}$. It is generally assumed that this mapping relation also holds for a new observation which is not present in $\boldsymbol{X}$, i.e.

$$\boldsymbol{u}_{new}^T = \boldsymbol{x}_{new}^T(\boldsymbol{DV}^T)^\dagger \tag{1}$$

given that $\boldsymbol{D}$ and $\boldsymbol{V}$ are regarded as constant. We can generalize these relationships to Matrix Factorization(MF) $\boldsymbol{X} = \boldsymbol{AB}^T$ as used in [16]. The latent embeddings are now rows of $\boldsymbol{A}$ and the weights as rows of $\boldsymbol{B}$. The mapping function now is

$$\boldsymbol{a}_{new}^T = \boldsymbol{x}_{new}^T(\boldsymbol{B}^T)^\dagger. \tag{2}$$

In both cases (SVD and MF), instead of a complete recalculation of the factorization to derive the corresponding latent embedding vector, we simply need to apply a linear map to $\boldsymbol{x}_{new}$, where the map is derived from the pseudo-inverse operation.

Tensor Cases: Following the notation in [15], we describe the CP/PARAFAC model [14] as well as its more general form, the Tucker decomposition [22], as $\boldsymbol{\mathcal{X}} \approx \boldsymbol{\mathcal{G}} \times_1 \boldsymbol{A} \times_2 \boldsymbol{B} \times_3 \boldsymbol{C}$. A row in each of the three matrices, i.e., $\boldsymbol{a}_{i,\bullet}, \boldsymbol{b}_{j,\bullet}, \boldsymbol{c}_{k,\bullet}$, stores the latent embedding of the i-, j- and k-th entity, respectively, in the corresponding dimensions of $\boldsymbol{\mathcal{X}}$; and the core tensor $\boldsymbol{\mathcal{G}}$ specifies the linear interaction between each triple of embedding vectors to derive the entry $x_{i,j,k}$. In the special case of CP, $\boldsymbol{\mathcal{G}}$ takes the form of a hyper-diagonal tensor. By rewriting the model as $\boldsymbol{X}_{(1)} = \boldsymbol{AG}_{(1)}(\boldsymbol{C} \otimes \boldsymbol{B})^T$ we could also interpret $\boldsymbol{A}$ as a latent embedding matrix and $\boldsymbol{G}_{(1)}(\boldsymbol{C} \otimes \boldsymbol{B})^T$ as the linear weights. Inverting this linear relation we can obtain a mapping of the form

$$\boldsymbol{a}_{new}^T = \boldsymbol{x}_{new}^T(\boldsymbol{G}_{(1)}(\boldsymbol{C} \otimes \boldsymbol{B})^T)^\dagger. \tag{3}$$

Such closed-form mappings cannot be derived in at least two cases: Firstly, for non-linear factorization models such as Multiway Neural Network (mwNN) [8], Neural Tensor Networks [20] and TransE [3]; secondly for models with shared embeddings such as RESCAL [19]. We derive solutions for those two cases in the next section. In the experimental part of this paper, we implement and test (logistic) MF to model data in matrix form and CP, RESCAL and mwNN for tensor data. A brief summary of the model architectures can be found in Table 1. To obtain proper probabilistic models, we introduce a natural parameter η for each entry x in the matrix or tensor.

3 General Models and Training Algorithms

In this section we introduce a generic framework describing factorization and **Em**bedding **Ma**pping (abbreviately termed 'Emma'). We also propose three

Table 1. A summary of selected factorization models within the scope of this paper. *sig* denotes the sigmoid or logistic function $sig(x) = \frac{1}{1+\exp(-x)}$; $\mathcal{N}$ and $\mathcal{B}$ denote Gaussian and Bernoulli distributions, respectively. We denote with MLP a standard three layered perceptron.

Model	Distr. assumption	Natural parameter
MF	$x_{i,j} \sim \mathcal{N}(\eta_{i,j}, \sigma)$	$\hat{\eta}_{i,j} = \sum_{r=1}^{R} a^{(1)}_{i,r} \cdot a^{(2)}_{j,r}$
Logistic MF	$x_{i,j} \sim \mathcal{B}(sig(\eta_{i,j}))$	
CP	$x_{i,j,k} \sim \mathcal{N}(\eta_{i,j,k}, \sigma)$	$\hat{\eta}_{i,j,k} = \sum_{r=1}^{R} a^{(1)}_{i,r} \cdot a^{(2)}_{j,r} \cdot a^{(3)}_{k,r}$
RESCAL	$x_{i,j,k} \sim \mathcal{N}(\eta_{i,j,k}, \sigma)$	$\hat{\eta}_{i,j,k} = \sum_{p}^{P} \sum_{p'}^{P} w_{p,p',k} \cdot a^{(1)}_{i,p} \cdot a^{(1)}_{j,p'}$
mwNN	$x_{i,j,k} \sim \mathcal{B}(sig(\eta_{i,j}))$	$\hat{\eta}_{i,j,k} = MLP([\boldsymbol{a}^{(1)}_{i,\bullet}, \boldsymbol{a}^{(2)}_{j,\bullet}, \boldsymbol{a}^{(3)}_{k,\bullet}]_+)$

novel approaches to train the mapping models. For the sake of generality we shall refer to matrices also as tensors.

3.1 General Models

The Factorization Model defines the interaction between latent embeddings to construct the tensor as

$$\begin{aligned} \widehat{\boldsymbol{\mathcal{H}}} &= f\left(\{\boldsymbol{A}^{(l)}\}_{l=1}^{L};\ \Theta\right) \quad \text{with} \\ \widehat{\boldsymbol{\mathcal{H}}} &\in \mathbb{R}^{n_1 \times \ldots \times n_L}, \boldsymbol{A}^{(l)} \in \mathbb{R}^{n_l \times p_l}. \end{aligned} \tag{4}$$

Here, the L-dimensional tensor $\boldsymbol{\mathcal{H}}$ contains the natural parameters η. The matrices in set $\{\boldsymbol{A}^{(l)}\}_{l=1}^{L}$ store the latent embeddings in their rows. The tensor is reconstructed with operations defined by a parameterized function $f(\bullet;\ \Theta)$.

For instance, in case of CP factorization, the function f specifies the linear combination of rows in each embedding matrix without additional parameters ($\Theta = \emptyset$); and for mwNN, Θ consists of the weights in an NN model whose architecture is defined as part of f.

The Factorization Cost Function: We define the factorization cost function c_F and its regularized version c_F^p as

$$c_F = d_F(\boldsymbol{\mathcal{X}}, \widehat{\boldsymbol{\mathcal{H}}}) = d_F(\boldsymbol{\mathcal{X}}, f(\{\boldsymbol{A}^{(l)}\}_{l=1}^{L};\ \Theta)) \tag{5}$$

$$c_F^p = c_F + \sum_{l=1}^{L} \gamma(\boldsymbol{A}^{(l)}) + \rho(\Theta). \tag{6}$$

During training the cost function is optimized w.r.t. the parameters in Θ and embeddings in the $\boldsymbol{A}^{(l)}$'s. An example for the differentiable distance measure d_F is the Frobenius Norm. For binary tensors, the cross-entropy loss is more suitable. In Eq. (6) we included a regularization function $\gamma(\bullet)$ for the latent embeddings and a second one $\rho(\bullet)$ for the model parameters.

The Mapping Model defines a function $g(\bullet;\ \Psi_l)$ that maps each row in the tensor unfolding $\boldsymbol{X}_{(l)}$ to the corresponding row in the learned embedding matrix $\boldsymbol{A}^{(l)}$ as

$$\begin{aligned} \widehat{\boldsymbol{A}}^{(l)} &= g\left(\boldsymbol{X}_{(l)};\ \Psi_l\right) \quad \forall l \in [1, ..., L] \quad \text{with} \\ \boldsymbol{X}_{(l)} &\in \mathbb{R}^{n_l \times \prod_{l' \neq l} n_{l'}}. \end{aligned} \tag{7}$$

Note that in the input of the mapping function, each arbitrary row i in $\boldsymbol{X}_{(l)}$, is identical to the vectorized i-th hyper-slice of the tensor and consists of all available information about the i-th entity. For a KG, for instance, this could be the vector indicating the existence of relations between entity i and all other entities for all relation types. The function $g(\bullet)$ defines the architecture of the mapping model and Ψ_l consists of all parameters.

The Mapping Cost Function: We define the mapping cost function as

$$c_M = \sum_{l=1}^{L} d_M(\boldsymbol{A}^{(l)}, \widehat{\boldsymbol{A}}^{(l)}) = \sum_{l=1}^{L} d_M(\boldsymbol{A}^{(l)}, g(\boldsymbol{X}_{(l)};\ \Psi_l)) \tag{8}$$

$$c_M^p = c_M + \sum_{l=1}^{L} \rho(\Psi_l). \tag{9}$$

Optimizing the mapping cost function involves adjusting Ψ_l for each l with a given $g(\bullet)$ so that the distance between the learned embedding $\boldsymbol{A}^{(l)}$ from factorization and mapped embedding $\widehat{\boldsymbol{A}}^{(l)}$ from the corresponding tensor unfoldings is minimized.

The Compact Model: Since the latent embeddings, e.g., the $\boldsymbol{A}^{(l)}$'s, are also adjustable parameters, we could write a more compact model by plugging Eq. (7) into Eq. (4) and obtain

$$\widehat{\boldsymbol{\mathcal{H}}} = f\left(\left\{g\left(\boldsymbol{X}_{(l)};\ \Psi_l\right)\right\}_{l=1}^{L};\ \Theta\right). \tag{10}$$

Analogously, combining cost functions of the factorization model —Eqs. (5) and (6)— and those of the mapping model —Eqs. (8) and (9)— we obtain:

The Compact Cost Function as:

$$c = d_F(\boldsymbol{\mathcal{X}}, \widehat{\boldsymbol{\mathcal{H}}}) = d_F\left(\boldsymbol{\mathcal{X}}, f\left(\left\{g\left(\boldsymbol{X}_{(l)};\ \Psi_l\right)\right\}_{l=1}^{L};\ \Theta\right)\right) \tag{11}$$

$$c^p = c + \rho(\Theta) + \sum_{l=1}^{L} \rho(\Psi_l), \tag{12}$$

where $\boldsymbol{A}^{(l)}$'s are not explicitly defined but could be derived from $g(\boldsymbol{X}_{(l)};\ \Psi_l)$.

It should be noted that the tensor $\boldsymbol{\mathcal{X}}$ occurs both at the output and the input (as unfoldings) of the cost function. More specifically, for a tensor $\boldsymbol{\mathcal{X}}$ of three dimensions, the factorization and mapping models as a whole actually model

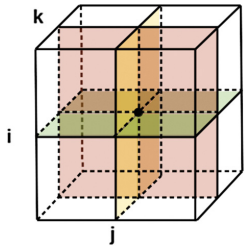


Fig. 1. Illustration of the Compact Model in case of a tensor: We model the each entry $x_{i,j,k}$ over the natural parameter $\eta_{i,j,k}$ based on the three slices of the tensor indexed by i, j and k, respectively.

each entry $x_{i,j,k}$ based on the i-th, j-th and k-th slices of 1st, 2nd and the 3rd dimension of the tensor, respectively, whereby the $x_{i,j,k}$ is the intersection of all these three slices. This aspect is illustrated in Fig. 1.

The factorization problem defined in Eq. (4) could be solved using a variety of well studied factorization models that optimize Eq. (5). In the following we focus on solving the mapping problem in Eq. (7) and the compact modelling problem in Eq. (10). Within the scope of this paper we specifically assume that the function $g(\bullet;\, \Psi_l)$ represents a linear relation between $\boldsymbol{A}^{(l)}$ and $\boldsymbol{X}_{(l)}$ that can be modelled by a matrix $\Psi_l = \{\boldsymbol{M}_l\}$, i.e.

$$\widehat{\boldsymbol{A}}^{(l)} = \boldsymbol{X}_{(l)}\boldsymbol{M}_l \;\forall l \in [1, ..., L]. \tag{13}$$

3.2 Training Approaches

Post Embedding Mapping: The most intuitive way to perform an Emma is to solve Eqs. (4) and (7) sequentially: Given that a certain factorization model has already derived the latent embeddings $\boldsymbol{A}^{(l)}$, we could consider the mapping from $\boldsymbol{X}_{(l)}$ to the embedding to be a linear system of n_l equations as suggested in Eq. (7). Since one is interested in small dimensions for the latent embeddings, i.e. $p_l < \prod_{l' \neq l} n_{l'}$, the system is overdetermined and can be approximately solved using Least-Square (LS) methods. Specifically: $\widehat{\boldsymbol{M}}_l = (\boldsymbol{X}_{(l)}^T \boldsymbol{X}_{(l)})^\dagger \boldsymbol{X}_{(l)}^T \boldsymbol{A}^{(l)}$.

It is easy to see that the inverting methods of Eqs. (1), (2) and (3) introduced in Sect. 2 are special cases of this LS estimation. For instance, with MF model $\boldsymbol{X} = \boldsymbol{A}\boldsymbol{B}^T$, the general LS estimation could be described as $\widehat{\boldsymbol{A}} = \boldsymbol{X}\boldsymbol{M}_A$ with $\boldsymbol{M}_A = (\boldsymbol{X}^T\boldsymbol{X})^\dagger \boldsymbol{X}^T\boldsymbol{A}$. Plugging in the information of the model definition, we obtain $\boldsymbol{M}_A = (\boldsymbol{B}\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{B}^T)^\dagger \boldsymbol{A}\boldsymbol{B}^T\boldsymbol{A} = (\boldsymbol{B}^T)^\dagger$, which is the same as the inverting operation in Eq. (2). An apparently desirable feature of this Post Mapping approach is its simplicity. It is applicable for any known factorization model and does not affect the factorizing process, since this approach assumes the learned embeddings as fixed. In the following we shall refer to this approach as Emma-Post.

Embedding Mapping Learning with Hatting Algorithm: Alternatively, one could also integrate the Emma learning into the factorization learning

Algorithm 1. Hatting Algorithm Framework

for all $l \in [1, ..., L]$ **do**:
  $\boldsymbol{U}^{(l)} \leftarrow (\boldsymbol{X}_{(l)}^T \boldsymbol{X}_{(l)} - \lambda \boldsymbol{I})^\dagger \boldsymbol{X}_{(l)}^T$
  $\boldsymbol{H}^{(l)} \leftarrow \boldsymbol{X}_{(l)} \boldsymbol{U}^{(l)}$
end for
for each epoch t in learning factorization **do**:
  $\{\boldsymbol{A}^{(l)}\}_{l=1}^L, \Theta \leftarrow$ update w.r.t. c_F^p

Absolute Updating:

for $l \in [1, ..., L]$ **do**:
  $\boldsymbol{A}^{(l)} \leftarrow \boldsymbol{H}^{(l)} \boldsymbol{A}^{(l)}$
end for

or

Stochastic Updating with Late-Starting:

if $t > \tau$ **then**:
  for $l \in [1, ..., L]$ **do**:
    $\pi^{(l)} = 1 - \max(0, R^2(\boldsymbol{A}^{(l)}, \boldsymbol{H}^{(l)} \boldsymbol{A}^{(l)}))$
    $\boldsymbol{A}^{(l)} \leftarrow \boldsymbol{H}^{(l)} \boldsymbol{A}^{(l)}$ with probability $\pi^{(l)}$
  end for
end if

end for
return:
  $\{\boldsymbol{A}^{(l)}\}_{l=1}^L$ as latent embeddings;
  $\{\boldsymbol{M}^{(l)} = \boldsymbol{U}^{(l)} \boldsymbol{A}^{(l)}\}_{l=1}^L$ as Emma matrices.

process: Instead of solving the LS problem after the factorization learning is completed, we suggest to fit the LS solution against the current latent embedding after each epoch of the factorization learning. Specifically, after each epoch of the factorization learning, we solve the LS problem based on the current embeddings and replace these with their LS estimates to satisfy the criterion of Eq. (8). In terms of notation, we replace $\boldsymbol{A}^{(l)}$ with $\widehat{\boldsymbol{A}}^{(l)} = \boldsymbol{H}^{(l)} \boldsymbol{A}^{(l)}$ i.e. we 'hat' the embedding matrix with the Hat-Matrix as in linear regression [13]. In the next epoch, the factorization algorithm proceeds from the LS estimates of the embeddings. In addition, to avoid collinearity and overfitting, it is also advisable to add a ridge regularization term to the Hat Matrix. We formulate this integrated Emma as an algorithmic framework in Algorithm 1 with the left-sided option termed *Absolute Updating*.

There are two major advantages of this algorithmic framework: Firstly, the LS updates are efficient to calculate since the Hat Matrix is only calculated once prior to the iterative factorization learning, during which one only needs to perform in each epoch one matrix multiplication for each dimension for the sake of LS-updating. Secondly, any factorization algorithm can be easily extended with this LS update as long as it is performed in a iterated manner, such as when using ALS or gradient based approaches.

Based on our experiments we also propose following practical adjustments of the algorithm:

i. Late Starting Strategy: Since the embeddings are usually initialized randomly by many algorithms, it is not necessary in such cases to perform LS updates during early epochs. It is advisable to start LS updates, for instance, when the embeddings are updated in smaller magnitudes i.e. where the cost

function is locally flat. One could measure the gradient changes in each epoch or simply define a $\tau > 1$ to be the first iteration where the LS update commences.

ii. Stochastic Update: We suggest monitoring the quality of the linear mapping in terms of R^2, the Coefficient of Determination. As long as R^2 is small, it is always assumed to be necessary to further perform LS update. But as the training proceeds, R^2 often tends to get larger and converges to 1. In such cases it may again be unnecessary to perform an LS update in each iteration and one could save some runtime. Since the R^2 typically lies within $(0, 1)$ but could also be negative based on our definition, we define a coefficient $\pi = 1 - \max(0, R^2)$ to be interpreted as the probability or necessity to perform an LS update. Both improvements are taken into account in the right-sided option *Stochastic Updating with Late-Starting* of Algorithm 1. In the following we shall refer to this approach as Emma-Hatting.

Embedding Mapping Learning with Back-Propagation: It is easy to see that, in combination with Multi-way Neural Networks, the linear mapping between tensor unfoldings and the latent embeddings could also be considered as one more linear activated layer of the network. To this end the Hatting Algorithm could be replaced with the usual Error Back-Propagation. This aspect also applies to other factorization models as long as such a model can be formulated as an NN. In such cases the latent embeddings become the first hidden layer and the tensor unfoldings become the input layer. For such models the latent embeddings are not explicitly learned but are derived from the product of the input vector and the mapping matrices. This aspect corresponds to the compact model described in Eq. (10) and is illustrated in Fig. 2. Similar illustrations could also be found in [18] for RESCAL and the mwNN. Note that for MF and RESCAL there are no parameters other than the mapping matrices to be learned; while the mwNN also learns the weights following the embedding vectors. In the following we shall refer to this approach as Emma-BP.

In summary, the Emma-Post approach optimizes the cost functions in Eqs. (5) and (8) consecutively and separately. Therefore the two error terms of $d_F(\boldsymbol{\mathcal{X}}, \widehat{\boldsymbol{\mathcal{H}}})$ and $d_M(\boldsymbol{A}^{(l)}, \widehat{\boldsymbol{A}}^{(l)})$ are —though minimized to a certain extent— always present. The Emma-Hatting approach also considers these two cost functions. But the LS estimates are calculated more than once and the LS error term —in the long run— is expected to be smaller than that derived from Emma-Post. However, because of this LS update within the factorization algorithm, the gradient approach may become unpredictable with respect to whether the optima identified with LS correction are better than those without the LS update. The stochastic Hatting Algorithm, from this point of view, could be considered as a compromise between the post mapping and the absolute hatting approach. On the one hand it still regulates the factorization algorithm to satisfy the cost function Eq. (8); on the other hand it allows the factorization algorithm to minimize the cost function of Eq. (5) continuously as long as the error of Eq. (8) is relatively small. In other words, this approach enables better factorization quality by tolerating some acceptable mapping error. By omitting

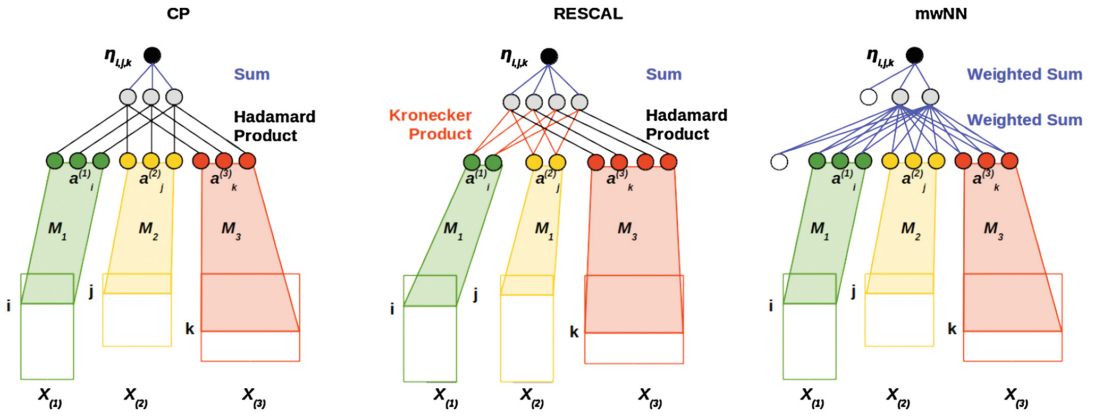


Fig. 2. Illustrating the Compact Model as NNs in 3-D case. Here the rows indexed by i, j and k in the tensor unfoldings $\boldsymbol{X}_{(1)}$, $\boldsymbol{X}_{(2)}$ and $\boldsymbol{X}_{(3)}$ correspond to the three coloured slices in Fig. 1, respectively. Note that for RESCAL there are only two mapping matrices instead of three since the entity embeddings are shared between subject and object. (Color figure online)

the explicit mapping error, the Emma-BP approach models the factorization and mapping as a whole only in terms of the factorization error of Eq. (11).

4 Experiments

In this section we present experiments on three real-world datasets and evaluate the results from two different aspects. First we evaluate our models on a user-item matrix and a KG dataset (both binary) in terms of prediction quality. Then, with another tensor dataset of real values, we focus on the interpretability of the mapped embeddings. The models were implemented in Keras [6] and its backend Theano [1].

4.1 Movielens Data

Data: The Movielens-100K [12] dataset is a user-item matrix containing 100000 ratings from 943 users on 1682 movies. The fact that a rating was performed on an item is encoded as a 1, otherwise we use a 0. Such binary-item matrix could be considered as a special case of a KG with two types of entities and one type of relation.

Task: The major task of a conventional recommender system is to predict the probability of a known user being interested in an known item, as long as event has not been observed in the past. (In terms of a KG this is equivalent to link prediction for one relation between two entities.) In contrast, we intend to predict the probabilities for a *new user* being interested in known items; or vice versa: for a new item to be of interest for known users.

Settings: First, we sample 20 % of users to hold out for test and train our models using the remaining 80 % with embedding sizes of 20 and 50. In the test phase,

we mask a sequence of proportions $[0, 0.1, 0.2, 0.3, 0.4, 0.5]$ of all watched movies of each test user and predict a distribution over all known movies, especially the masked ones. We measure the quality of each prediction in terms of NDCG@k [5]. Further we transpose the user-movie matrix and conduct the same experiment, i.e., we add new movies to the database and try to recommend each movie to the most likely user. Other than [11], we test a logistic MF combined with Emma-Post, Emma-Hatting and Emma-BP. As baseline model we perform a most-popular prediction: we calculate the frequencies of each movie for all users in the training set and interpret these as constant recommendation scores for a new user and a new movie, respectively. We repeat this process 5 times with different and mutually exclusive training and test samples in order to derive prediction stability in terms of mean and standard deviations.

Results: The results of the experiments are presented in Fig. 3. The mean and standard deviation of the NDCG@k scores of the three mapping techniques are visualized in corresponding colors. The horizontal axis represents the proportion of masked entries in the test set; the mean and standard deviation of baseline predictions are demonstrated as the horizontal line and the gray bars.

The performances of the models suggest different trends for new-user and new-movie cases. For new users, the predictions made by logistic MF with Emma-Hatting are suboptimal and may even drop below the baseline for larger masking proportions. Emma-Post and Emma-BP offer better and comparable prediction qualities, though the latter remains advantageous even as the masking proportion becomes quite large. Emma-Post, however, cannot even beat the baseline for new appended movies, while Emma-BP perform noticeably well for all possible masking proportions.

4.2 Knowledge Graph Data

Data: In following experiments we test our models on the Freebase dataset [2] as prepared in [17]. We sample entities that contain at least 500 known relations forming a binary tensor of shape $39 \times 115 \times 115$.

Task: Similar to recommender systems, the conventional link prediction in KGs is performed for each triple of known entities and relations [21]. With our Emma models we intend to predict the existence of links between known and new entities, given some observed but incomplete information about this new entity. More specifically, for an existing KG modelled with a binary tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times I \times K}$ we assume a further binary tensor $\boldsymbol{\mathcal{Z}} \in \mathbb{R}^{I' \times I \times K}$ storing a subset of true links between I' new entities and the I known ones. Our task here shall be to predict the unobserved links in $\boldsymbol{\mathcal{Z}}$ based on factorization and mapping models trained only on $\boldsymbol{\mathcal{X}}$.

Settings: In order to also estimate the model's stability we perform a 20 %−80 % Cross-Validation on the data by splitting the entity set into 5 mutually exclusive groups. In each test set we mask and try to recover 20 % of known links for each entity with two approaches: (1) We map the entities with masked links into

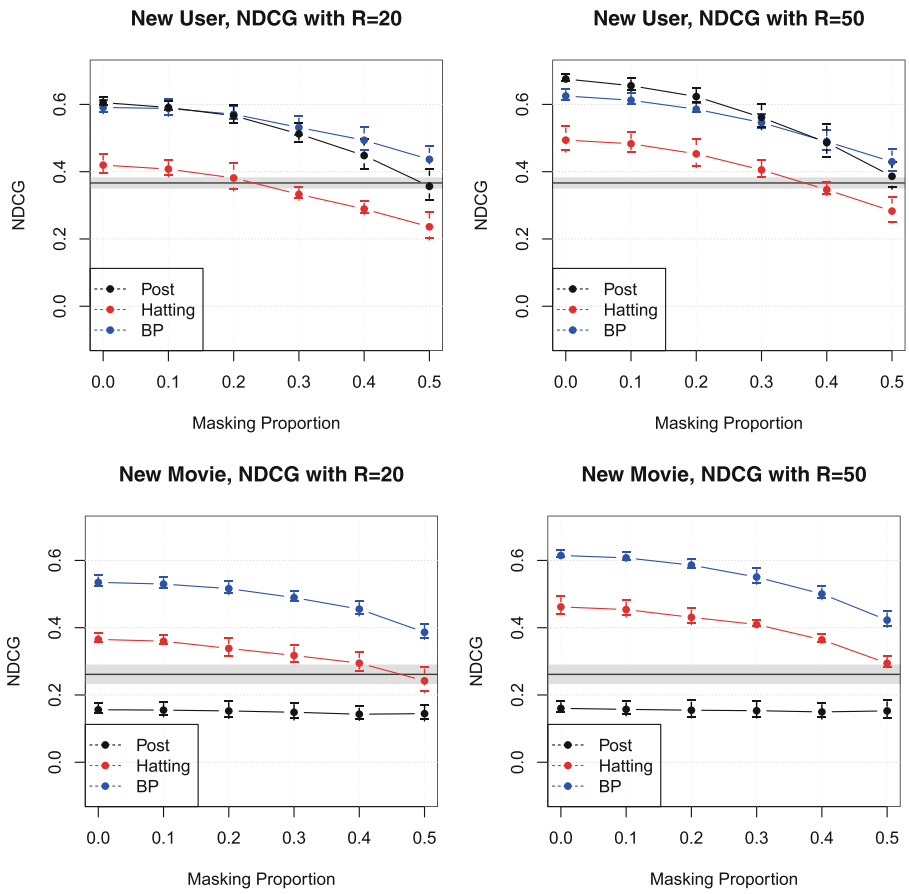


Fig. 3. Evaluation of prediction for new users and movies. The horizontal line and grey bar represent the baseline recommendation and its standard deviation. (Color figure online)

the latent space obtained by the Emma-Post, Emma-Hatting and Emma-BP models that have been trained with the corresponding training set and predict the masked links with the same models; (2) we train a RESCAL and a mwNN model on training *and* test sets, simulating a retraining scenario, and predict the masked links in the conventional way such as in [17]. In both cases we generate negative samples according to the Local-Closed-World-Assumption [17].

Results: In Table 2 we report the prediction quality of AUROC and AUPRC using models of RESCAL and mwNN in combination with all three Emma approaches as well as from retraining. In general, mwNN outperforms RESCAL in terms of larger means and smaller standard deviations in almost all cases, which could also be supported by the results reported in [17]. In predicting for new entities, mwNN combined with Emma-BP yields the best mean values. Especially in terms of AUPRC the advantage could be as large as 33 % compared to second best result produced by RESCAL+Emma-Hatting for $R = 10$ and 44 % compared with RESCAL+Emma-BP for $R = 30$. The minimal standard deviations are achieved in 5/8 cases by Emma-Post, though it almost always produces worst mean values in combination with any factorization models. As

Table 2. Prediction Qualities of RESCAL and mwNN in combination with all three mapping approaches on a FreeBase dataset.

Fac.	Mapping	R = 10		R = 30	
		AUROC	AUPRC	AUROC	AUPRC
RESCAL	Retraining	0.901 ± 0.039	0.820 ± 0.059	0.788 ± 0.054	0.616 ± 0.100
	Emma-Post	0.759 ± 0.096	0.600 ± 0.145	$0.693 \pm \mathbf{0.065}$	$0.432 \pm \mathbf{0.106}$
	Emma-Hatting	$\mathbf{0.778} \pm 0.112$	$\mathbf{0.605} \pm 0.152$	0.700 ± 0.091	0.481 ± 0.120
	Emma-BP	$0.700 \pm \mathbf{0.060}$	$0.485 \pm \mathbf{0.092}$	$\mathbf{0.740} \pm 0.090$	$\mathbf{0.509} \pm 0.134$
mwNN	Retraining	0.964 ± 0.008	0.886 ± 0.060	0.970 ± 0.010	0.923 ± 0.017
	Emma-Post	$0.844 \pm \mathbf{0.009}$	0.390 ± 0.101	$0.826 \pm \mathbf{0.035}$	$0.382 \pm \mathbf{0.063}$
	Emma-Hatting	0.847 ± 0.042	0.423 ± 0.118	0.843 ± 0.038	0.394 ± 0.081
	Emma-BP	$\mathbf{0.949} \pm 0.022$	$\mathbf{0.805} \pm \mathbf{0.080}$	$\mathbf{0.931} \pm 0.036$	$\mathbf{0.735} \pm 0.101$

expected, retraining always offers better predictions than Emma approaches. But do note that a prediction with an Emma model does not cost any run time; while a retraining process for one or multiple entities would demand a comparable amount of time as training an Emma model from scratch. Interpreting the retraining predictions as upper bound, it should also be noted that the combination of mwNN and Emma-BP achieves in most cases results relatively close to those of retraining. We speculate that such canonical model-algorithm combination might enjoy numerical advantage.

4.3 Amino Acid Data

With previous experiments we have shown that Emma models are able to predict links between every known entity and a newly appended entity based on incomplete information. With the following experiment we also show that Emma models can map a new entity into the latent space with high interpretability — here in terms of correlation coefficients.

Data: The Amino Acid Dataset [4] contains a three-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{5\times51\times201}$ and a matrix $\boldsymbol{Y} \in \mathbb{R}^{5\times3}$. The latter one describes the proportion of 3 types of amino acid mixed according to 5 different recipes. The corresponding 5 samples are then measured by fluorescence with excitation 250–300 nm, emission 250–450 nm on a spectrofluorometer and the measurements are recorded in the tensor $\boldsymbol{\mathcal{X}}$. With a CP factorization producing matrices of dimensions $\boldsymbol{A}^{(1)} \in \mathbb{R}^{5\times3}$, $\boldsymbol{A}^{(2)} \in \mathbb{R}^{201\times3}$ and $\boldsymbol{A}^{(3)} \in \mathbb{R}^{61\times3}$ it is expected that each column in $\boldsymbol{A}^{(1)}$ would strongly correlate with one column in the recipe matrix $\boldsymbol{Y}$. Note that the order of the columns in $\boldsymbol{A}^{(1)}$ is arbitrary and may not correspond to the column in $\boldsymbol{Y}$ at the same position. For more details please refer to [4].

Task: The latent embeddings learned from this data set are expected to be interpretable in terms of correlations with known recipes. If a new entity is correctly mapped into the latent space, its *mapped* embedding(s) along with other *learned* embeddings would also correlate with the corresponding column

in the recipe matrix. Here we assume the information observed on the new entity to be complete and do not perform any masking.

Settings: We remove each slice $\boldsymbol{X}_{i,\bullet,\bullet}, i \in [1,5]$ (corresponding to a certain recipe) from the tensor and calculate CP factorizations with Emma-Post and Emma-Hatting based on the rest of the data. The slice held out is then mapped to the 3-D latent space with mapping matrix and appended to the learned embeddings of the other slices. We then calculate the mean of its column-wise Pearson correlation coefficients with $\boldsymbol{Y}$. Further we conducted the same experiment with two slices removed at a time.

Results: With the first leaving-one-out experiment setting we derive accordingly 5 averaged correlations and for the second leaving-two-out setting we have $\frac{5!}{3!\cdot 2!} = 10$ values. We report that the means and standard deviations of the correlation coefficients derived from Emma-Post and Emma-Hatting to be both 0.999 ± 0.001. As for leaving-two-out experiments, Emma-Post achieves 0.991 ± 0.007 and Emma-Hatting yields the same mean value but a larger standard deviation of 0.015. To this end we conclude that both mapping approaches can map one or two new slices into the latent space in a way that its embedding(s) —along with other embeddings learned from factorization— correlates column-wise with the recipe matrix $\boldsymbol{Y}$ with a high correlation score. As expected, in leaving-two-out experiments the correlation coefficients decrease slightly due to the smaller training set. Note that once again Emma-Post seems to produce smaller standard deviations just as the KG experiments.

5 Related Works

[10] introduced a method to map user attributes (referred to as 'content information' in the context of content filtering) into the latent embedding space to solve cold-start problem for new entities. Despite the fact that we are not considering the cold-start problem and do not require content information, this model still shares a few aspects with ours: (1) In both approaches, the codomain of the mapping function is the latent space learned by factorization models with the purpose of finding latent embeddings for new entities; (2) both approaches use modular learning that can be combined with a variety of existing factorization models. An important difference is the domain of the mapping function. In our case it is the observable feature space of an entity, while in [10], it is the user attribute space. It would be interesting, though, to combine our models with such content information: In the first step, one could perform content filtering to produce some first recommendations. Secondly, one could interpret these recommendations as incomplete and enrich them using Emma models by mapping them into the latent space and perform link predictions.

Our proposed Emma-BP model shares the idea of learning the factorization jointly with an implicit latent embeddings with the Temporal Latent Embedding (TLEM) Model [9], where a concatenated NN is trained with observable features as inputs, which are mapped to some implicit latent features. In TLEM one

intends to model the *consecutive* effect of a sequence of events on the next one; while we are interested in the *collaborative* effect among entities.

6 Conclusions

The major contribution of this paper is to propose three approaches for mapping new entities into the latent embedding space learned by a wide variety of factorization models.

Our approaches are not based on retraining while obtaining comparable quality to model retraining. Our framework describes factorization and mapping problems on an abstract level, which could inspire the development of further mapping approaches. During our experiments we also realized the model's restrictions in practice: Due to the unfolding operations in the mapping model, the dimensions of the model inputs increase quadratically with the dimensions of the tensor. For instance, in our KG experiment in Sect. 4 with a tensor of shape $39 \times 115 \times 115$, the mapping model requires inputs of dimensions 4485, 4485 and 13225, respectively. As part of future work, we will explore different approaches for dimensionality reduction. In addition, we plan to study non-linear extensions to Emma as well as the application to recurrent NNs.

References

1. Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., Bengio, Y.: Theano: a CPU and GPU math expression compiler. In: Proceedings of the SciPy (2010)
2. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database. In: ACM SIGMOD (2008)
3. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: NIPS (2013)
4. Bro, R.: Multi-way analysis in the food industry: models, algorithms, and applications. Ph.D. thesis, Københavns Universitet (1998)
5. Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G.: Learning to rank using gradient descent. In: Proceedings of the 22nd International Conference on Machine Learning, pp. 89–96. ACM (2005)
6. Chollet, F.: Keras: deep learning library for theano and tensorflow (2015). https://github.com/fchollet/keras
7. Culotta, A., McCallum, A.: Joint deduplication of multiple record types in relational data. In: ACM Information and Knowledge Management (2005)
8. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., Zhang, W.: Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In: ACM SIGKDD (2014)
9. Esteban, C., Schmidt, D., Krompaß, D., Tresp, V.: Predicting sequences of clinical events by using a personalized temporal latent embedding model. In: ICHI (2015)
10. Gantner, Z., Drumond, L., Freudenthaler, C., Rendle, S., Schmidt-Thieme, L.: Learning attribute-to-feature mappings for cold-start recommendations. In: ICDM (2010)

11. Gantner, Z., Rendle, S., Freudenthaler, C., Schmidt-Thieme, L.: Mymedialite: a free recommender system library. In: ACM Conference on Recommender Systems (2011)
12. Herlocker, J.L., Konstan, J.A., Borchers, A., Riedl, J.: An algorithmic framework for performing collaborative filtering. In: ACM SIGIR (1999)
13. Hoaglin, D.C., Welsch, R.E.: The hat matrix in regression and anova. Am. Stat. **32**(1), 17–22 (1978)
14. Kiers, H.A.: Towards a standardized notation and terminology in multiway analysis. J. Chemometr. **14**(3), 105–122 (2000)
15. Kolda, T.G., Bader, B.W.: Tensor decompositions and applications. SIAM Rev. **51**, 455–500 (2009)
16. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
17. Krompaß, D., Baier, S., Tresp, V.: Type-constrained representation learning in knowledge graphs. In: ISWC (2015)
18. Nickel, M., Murphy, K., Tresp, V., Gabrilovich, E.: A review of relational machine learning for knowledge graphs. Proc. IEEE **104**(1), 11–33 (2016)
19. Nickel, M., Tresp, V., Kriegel, H.P.: A three-way model for collective learning on multi-relational data. In: ICML (2011)
20. Socher, R., Chen, D., Manning, C.D., Ng, A.: Reasoning with neural tensor networks for knowledge base completion. In: NIPS (2013)
21. Taskar, B., Wong, M.F., Abbeel, P., Koller, D.: Link prediction in relational data. In: NIPS (2003)
22. Tucker, L.R.: Some mathematical notes on three-mode factor analysis. Psychometrika **31**(3), 279–311 (1966)

Chapter 3

RNNs in Sequential EHR for Predictive Decision Support

Predictive Modeling of Therapy Decisions in Metastatic Breast Cancer with Recurrent Neural Network Encoder and Multinomial Hierarchical Regression Decoder

Yinchong Yang, Volker Tresp
Ludwig-Maximilians-Universität München
Siemens AG, Corporate Technology, Munich
{*yinchong.yang, volker.tresp*}*@siemens.com*

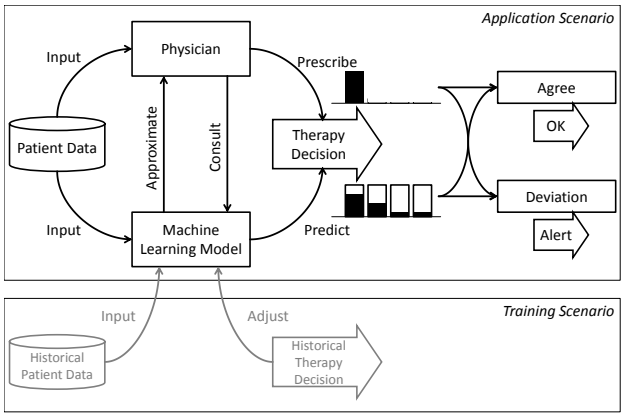
Peter A. Fasching
Department of Gynecology and Obstetrics,
University Hospital Erlangen
peter.fasching@uk-erlangen.de

***Abstract*—The increasing availability of novel health-related data sources —e.g., from molecular analysis, health Apps and electronic health records— might eventually overwhelm the physician, and the community is investigating analytics approaches that might be useful to support clinical decisions. In particular, the success of the latest developments in Deep Learning has demonstrated that machine learning models are capable of handling —and actually profiting from— high dimensional and possibly sequential data. In this work, we propose an encoder-decoder network approach to model the physician's therapy decisions. Our approach also provides physicians with a list of similar historical patient cases to support the recommended decisions. By using a combination of a Recurrent Neural Network Encoder and a Multinomial Hierarchical Regression Decoder, we specifically tackle two common challenges in modeling clinical data: First, the issue of handling episodic data of variable lengths and, second, the need to represent hierarchical decision procedures. We conduct experiments on a large real-world dataset collected from thousands of metastatic breast cancer patients and show that our model outperforms more traditional approaches.**

1. Introduction

With the introduction of the Electronic Health Records (EHR), a large amount of digital information has become available in clinics. This is expected to encourage more personal and precise healthcare services and improve patients experience [1, 2]. On the other hand, it also requires the physicians to consult a large variety and volume of data in order to perform diagnosis and treatment decisions, such as the patients' background information, medical images, genetic profiles and the patients' entire medical history. The decision making process, therefore, could become increasingly complex in connection with the growing amounts of information collected on each patient. Machine learning based Clinical Decision Support (CDS) systems could provide a solution to such data challenges [3, 4, 5]. These systems are able to actually profit from the large amount of data in high dimensional space. For instance, the latest success of Deep Learning models lies in their ability to generate more abstract and informative latent features from the high dimensional raw features, which turns out to largely facilitate predictive modeling.

Figure 1. The concept of a machine learning based CDS system.



There are multiple ways that a machine learning model may impact the decision process of a physician, for instance, by predicting the possible outcome of each decision. [5] provides physicians with endpoint predictions of patients with kidney failure. Based on the predicted probabilities of kidney rejection, kidney loss and death within the next 6 and 12 months, the physician is more informed to select the correct medication. This class of approaches, however, might be limited when i) not yet enough endpoints are labeled in the training data and ii) confounder effect is presumed in the data situation [6]. Therefore, in this work we explore another approach to machine learning base decision support by directly predicting the physicians' decisions. More specifically, a machine learning model would calculate the probability of each decision conditioned on the patient information. From the viewpoint of the physicians, these probabilities can be interpreted as recommendation scores. We illustrate this conceptual framework in Fig. 1. When properly trained, the machine learning model can be expected

to generate recommendations which –to a certain extend– agree with the prescriptions actually prescribed by the committee of physician in the study. In cases where the physician faces a great number of possible decisions, the recommendations would narrow down the size of prescription candidates. On the other hand, the machine learning model would also implicitly detect anomalous prescriptions, by checking whether the actual prescriptions are among the top-n ranked recommendations made by the machine learning model. Such a system relies on the predictive power of the machine learning model, which can be trained using historical data. During training, the model attempts to predict historical decisions based on the corresponding patient data and the actually documented decisions can adjust the model so that it can improve its predictions throughout the training epochs.

Our study is based on a large and up-to-date data set consisting of almost three thousand metastatic breast cancer patients in Germany. This dataset includes the patients' background information, the primary tumor record, etc., as well as the development of the cancer such as local recurrence and metastasis. Included in the dataset are also all the prescribed treatments each patient obtained throughout time. Since the physicians make their therapy decisions — often at a tumor board— after studying all available patient information, we assume that a machine learning model can also be trained to map the patient features to the therapy decisions.

There are two major challenges in the data situation. Firstly, the patients in the dataset do not share a time axis and do not visit clinics regularly. On some patients, no more data was ever recorded after a surgery; while others revisited the clinics repeatedly for years due to local recurrences and metastasis. Consequently, the patients have a medical history of variable length, making it challenging to construct a common input feature space for all patients. Secondly, the therapy decisions that we attempt to model are of a hierarchical structure. For instance, the physician first has to decide for a radiotherapy before further specifying whether it should be a curative or a palliative one, and whether it should be a Brachytherapy or a percutaneous type.

To address these two issues we propose a neural network architecture that instantiates the Encoder-Decoder Framework by [7]. Specifically, we encode the patients' medical histories of *variable lengths* into one *fixed-size* representation vector using Recurrent Neural Networks (RNN), and deploy on top of that a hierarchical regression model, which functions as a decoder that predicts the therapy decisions. We conduct experiments on the dataset with multiple choices of encoders and decoders, as well as different hyper-parameter settings, and identify their contribution to the modeling quality. Furthermore, we show that with our model architecture, one could also provide physicians with a list of similar historical patient cases to support our prediction, making it more realistic to deploy such decision support system in clinics.

The rest of the paper is organized as follows. In Section 2 we discuss multiple related works that inspired the design of our model. In Section 3 we describe our data situation, including the study background and data processing approaches. In Section 4 we first briefly introduce two specific RNN models that serve as our encoder network, and then propose a hierarchical prediction model as our decoder. In Section 5 we present our experimental results and Section 6 wraps up our present work and give a outlook for future directions.

2. Related Work

Handling sequential EHR data. Due to the sequential nature of EHR data, there have recently been multiple promising works studying clinical events as sequential data. Many of them are inspired by works in natural language modeling, since sentences can be easily modeled as sequence of signals. [4] adjusted a language model based on the sliding window technique in [8], taking into account a fixed number of events in the past. This model was extended in [5] by replacing the sliding window with RNNs, which improved the predictions for prescriptions decision and endpoints. [9] also applied LSTM-RNN to perform diagnosis prediction based on sequential input. And a related approach with RNNs can also be found in [3] to predict diagnosis and medication prescriptions at the same time throughout time. Such RNN application was further augmented with neural attention mechanism in [10], which did not only show promising results but also improved the interpretability of the model.

RNNs for sequence classification/regression. The RNN models in these works were implemented in a many-to-many fashion. That is to say, at each time step the RNN is supposed to generate a prediction as output. The reason is that in their data all patients share the same aligned time axis and regularly visit the clinics. In our work, on the other hand, there are neither regular visits nor shared time axis. To this end we implemented many-to-one RNN models that consume a sequential input and generates only one output. This setting can be found in a variety of sequence classification/regression tasks. [11] used such RNN architectures to classify spoken words and handwriting as sequences. RNNs have been also applied to classify the sentiment of a sentence such as in the IMDB reviews dataset [12]. The applications of the RNNs in the many-to-one fashion can also be seen as the encoding of a sequence of variable length into one fixed-size representation [7], which is then decoded to perform prediction as decoding.

Hierarchical classification/regression model. Rather than a simple classification task where all classes are on the same level, the therapy decisions turn out to be more complicated. For instance, the decision of a Brachytherapy is only observed when the physician decides to prescribe a radiotherapy in the first place. In order to model such a decision procedure as realistic as possible, we extend a hierarchical response model in [13] and deploy it as decoder on top of RNNs. [14] also proposed a quite similar architecture to factorize a large softmax layer into a hierarchy. The purpose was to accelerate the calculation of the softmax, which in natural language processing often has the size of the entire vocabulary.

3. Metastatic Breast Cancer Data

In this section we first briefly introduce the classical breast cancer therapies and then give an overview of our data situation.

3.1. Metastatic Breast Cancer Treatments

Breast cancer is the one of the most frequent malignant cancers in the Western world. In Germany, for instance, approximately 70,000 women suffer from breast cancer each year with around 30% mortality rate [15, 16]. In many of these cases, it is the metastasis of the cancer cells to vital organs that actually causes the patient's death. There are three classes of classical treatments of metastatic breast cancer: radiotherapy, systemic therapy and surgery. Typically, as soon as a patient is diagnosed with breast cancer, a surgery to remove the primary tumor would be the first option. In order to prevent local recurrence and metastasis, the patient would receive radiotherapies and/or systemic therapies after the surgery. If, however, local recurrences and/or metastasis are later diagnosed, the patient might undergo a further surgery, succeeded by radiotherapies and/or systemic therapies. This process can be repeated till either i) no more recurrence or metastasis can be identified or ii) the metastasis is observed in vital organs and surgery is no longer an option. In the latter case, radiotherapies and/or systemic therapies would become the major treatments. Latest discoveries in genetics have brought about novel systemic therapies that exploit specific biological characteristics of the cancer cell. Since these special characteristics are mostly not present in healthy cells, these targeted therapies have proven to be more efficient with less severe adverse effect.

3.2. Data Description and Processing

The majority of the dataset was provided by the PRAEGNANT study network [17], which has been recruiting patients of metastatic breast cancer since 2014. The original data are warehoused in the secuTrial® database. After exporting and pre-processing, we could extract information on 2,869 valid patients.

There are two classes of patient information that are potentially relevant for modeling the therapy decisions: First the *static* information includes 1) basic patient properties, 2) information on the primary tumor and 3) information on the history of metastasis before entering the study. In total we observe 26 features of binary, categorical or real types. We performed dummy-coding on the former both cases and could extract for each patient i a static feature vector denoted with $\boldsymbol{m}_i \in \mathbb{R}^{118}$. We summarize the features in Tab. 1.

The *sequential* information includes data on 4) local recurrences, 5) metastasis 6) clinical visits 7) radiotherapies, 8) systemic therapies and 9) surgeries. These are time-stamped clinical events observed on each patient throughout time, and at each time step there can be more than one type of events recorded. All these sequential features are of binary or categorical nature and are also dummy-coded, yielding

Table 1. OVERVIEW OF ALL STATIC FEATURES.

Static features	Feature names and dimensions	
1) Basic	Age	1
	Height	1
	HRT (Hormone Replacement Therapy)	5
	parity	9
	Mother BC	3
	Sister BC	6
	Menstruation	1
2) Primary Tumor	Type	3
	Total eval. of the malignancy	8
	Total eval. of axilla	4
	TAST eval. of the malignancy	8
	TAST eval. of axilla	4
	Mammography eval. of the malignancy	8
	Mammography eval. of axilla	4
	Ultrasound eval. of the malignancy	8
	Ultrasound eval. of axilla	4
	MRI eval. of the malignancy	8
	MRI eval. of axilla	8
	Metastasis staging	4
	Ever neoadjuvant therapy	4
	Ever surgery	4
3) History of metastasis	Lungs	1
	Liver	1
	Bones	1
	Brain	1
	Other	10
Total	26	118

for patient i at time step t a feature vector $\boldsymbol{x}_i^{[t]} \in \{0,1\}^{189}$. We denote the whole sequence of events for this patient i up to time T_i using a set of $\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i}$. We summarize the sequential features in Tab. 2.

Since we attempt to model the therapy decisions concerning radiotherapies (item 7), systemic therapies (item 8)[1] and surgeries (item 9), we extract from the medical history of each patient all possible sub-sequences where the last event consists of one of the three therapies. Therefore in each of these sub-sequences, the last event serves as the target that the model is expected to predict based on all previous events and the static information. Obviously, instead of the entire vector $\boldsymbol{x}_i^{[t]}$ we only need the subset of the vector concerning the therapies and denote this with $\boldsymbol{y} \in \{0,1\}^{39}$. Finally the training/test samples are constructed as

$$\{\boldsymbol{m}_i, \{\boldsymbol{x}_i^{[t]}\}_{t=1}^{t_i^*-1}\} \rightarrow \boldsymbol{y}_i^{[t_i^*]} \subseteq \boldsymbol{x}_i^{[t_i^*]}, \tag{1}$$

for each possible time step t_i^* where one of the therapies is observed. We illustrate this approach of data processing in Fig. 2.

From the 2,869 patients we could extract in total 16,314 sequences (i.e. 5.7 sequence per patient on average). The length of the sequence before a therapy prescription varies from 0 to 35 and is on average 4.1.

Every time a physician is supposed to prescribe a treatment, she/he is first supposed to choose one of the three *therapy categories* of radiotherapy, systemic therapy and surgery. For each chosen therapy category the physician

1. Except the fourth feature "Reason of termination", which we do not deem predictable but would serve well as input feature.

Table 2. OVERVIEW OF ALL SEQUENTIAL FEATURES, THEREOF 7), 8) AND 9) ARE THERAPIES THAT WE ATTEMPT TO PREDICT.

Sequential features	Feature names and dimensions	
4) Local Recurrences	Location	4
	Type	3
5) Metastasis Evaluation	Total	6
	Lungs	9
	Liver	9
	Bones	9
	Brain	9
	Lymph	9
	Skin	9
	Ovum	9
	Soft tissue	8
	Kidney	8
	Pleural cavity	8
	Thorax	8
	Muscle	8
	Periosteum	8
	Other	8
6) Visits	Therapy situation	12
	ECOG Life status	6
7) Radiation	Type	3
	Intention	3
8) Systemic	Type	6
	Intention	13
	Ref. to an surgery	4
	Reason of termination	6
9) Surgery	Type	10
Total	26	189

will then decide the *therapy features*. For radiotherapy there are two 3-dimensional multinomial distributed features: the radiotherapy intention being either curative, palliative or unknown; and the radiotherapy's type being either percutaneous, Brachytherapy or others. For systemic therapy there are three multinomial distributed features. The first one describes 6 types of systemic therapy such as antihormone therapy, chemotherapy, anti-HER2 therapy etc.; the second feature documents the therapy's intention, namely an argument based on the 13 different stagings of the cancer; the third four-dimensional feature records whether the therapy prescription is related to a surgery or is unknown. The last category is composed of 10 Bernoulli distributed variables that describe the surgery, such as breast conservation surgery, mastectomy, etc.. Detailed information of the feature *values* can be found in Tab. 3.

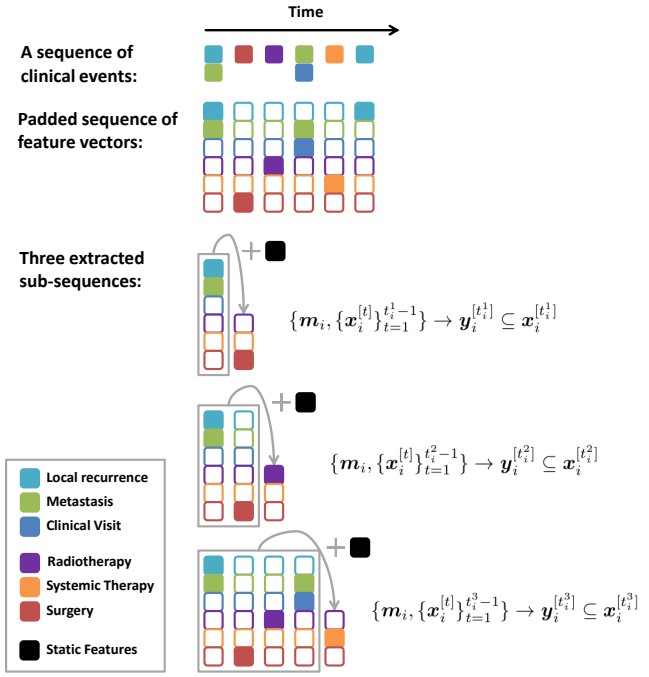
4. A Predictive Model of Therapy Decisions

In this section we provide an introduction to the two core ingredients of our proposed model: the many-to-one RNNs and a Multinomial Hierarchical Regression model. Eventually, both will be joined to form the complete predictive model.

4.1. Recurrent Neural Network as Encoder

Recurrent Neural Networks, especially the more advanced variants of Gated Recurrent Units (GRU) [18], presented in Eq. (2), and Long Short-Term Memory (LSTM) [19, 20] as in Eq. (3) have proven to be powerful in modeling

Figure 2. Illustration of generating training and test sequences from the medical history of a patient. From a complete sequence of clinical events, we extract all possible sub-sequences that end with one or multiple therapies, in this case at t_i^1, t_i^2 and t_i^3. At each time step, if a specific event is not observed, its corresponding features are zero-padded, yielding a common feature space at each time step.



multidimensional sequential data such as sensory and natural language data [21, 22].

GRU:

$$\begin{aligned} \boldsymbol{r}^{[t]} &= \sigma(\boldsymbol{W}^r \boldsymbol{x}^{[t]} + \boldsymbol{U}^r \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^r) \\ \boldsymbol{z}^{[t]} &= \sigma(\boldsymbol{W}^z \boldsymbol{x}^{[t]} + \boldsymbol{U}^z \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^z) \\ \boldsymbol{d}^{[t]} &= \tanh(\boldsymbol{W}^d \boldsymbol{x}^{[t]} + \boldsymbol{U}^d (\boldsymbol{r}^{[t]} \circ \boldsymbol{h}^{[t-1]})) \\ \boldsymbol{h}^{[t]} &= (1 - \boldsymbol{z}^{[t]}) \circ \boldsymbol{h}^{[t-1]} + \boldsymbol{z}^{[t]} \circ \boldsymbol{d}^{[t]}, \end{aligned} \tag{2}$$

LSTM:

$$\begin{aligned} \boldsymbol{k}^{[t]} &= \sigma(\boldsymbol{W}^k \boldsymbol{x}^{[t]} + \boldsymbol{U}^k \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^k) \\ \boldsymbol{f}^{[t]} &= \sigma(\boldsymbol{W}^f \boldsymbol{x}^{[t]} + \boldsymbol{U}^f \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^f) \\ \boldsymbol{o}^{[t]} &= \sigma(\boldsymbol{W}^o \boldsymbol{x}^{[t]} + \boldsymbol{U}^o \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^o) \\ \boldsymbol{g}^{[t]} &= \tanh(\boldsymbol{W}^g \boldsymbol{x}^{[t]} + \boldsymbol{U}^g \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^g) \\ \boldsymbol{c}^{[t]} &= \boldsymbol{f}^{[t]} \circ \boldsymbol{c}^{[t-1]} + \boldsymbol{k}^{[t]} \circ \boldsymbol{g}^{[t]} \\ \boldsymbol{h}^{[t]} &= \boldsymbol{o}^{[t]} \circ \tanh(\boldsymbol{c}^{[t]}). \end{aligned} \tag{3}$$

Both models generate for each time stamped input $\boldsymbol{x}^{[t]}$ a hidden state $\boldsymbol{h}^{[t]}$ that depends on both the current input $\boldsymbol{x}^{[t]}$ and the last representation $\boldsymbol{h}^{[t-1]}$.

If one has a sequence of targets $\{\boldsymbol{y}^{[t]}\}_{t=1}^{T}$ with the same length as $\{\boldsymbol{x}^{[t]}\}_{t=1}^{T}$ (a many-to-many model) such as in [3, 5], one could build a prediction model on top of every hidden state: $\hat{\boldsymbol{y}}^{[t]} = \phi(\boldsymbol{h}^{[t]})\ \forall t$. On the other hand, one could also

Table 3. THE MODELING TARGET: THERAPY CATEGORIES, THERAPY FEATURES AND FEATURE VALUES.

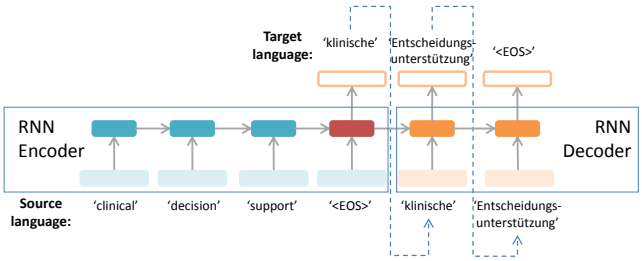
Therapy category	Therapy feature	Feature value
7) Radiation	Intention	Curative Palliative unknown
	Type	percutaneous radiation Brachytherapy Other radiotherapies
8) Systemic	Type	Anti-hormone therapy Chemotherapy Anti-HER2 therapy Other anti-body therapy Bone specific therapy Other therapies
	Intention	CM0/First treatment. CM0/Treatment of local recc. 1st line met 2nd line met 3rd line met 4th line met 5th line met 6th line met 7th line met 8th line met 9th line met not filled unknown
	Ref. to surgery	Neoadjuvant Adjuvant No surgery Unknown
9) Surgery	Type	Breast-Conserving Therapy Mastectomy Excision Trial Sampling Diagnostic Sampling Sentinel-Node-Biopsy Skin Sparing Mastectomy Port-Implantation Paracentesis Reconstruction

have a many-to-one model with only one target y for the whole input sequence of $\{\boldsymbol{x}^{[t]}\}_{t=1}^{T}$. In such case a prediction model consumes the last hidden state, which recurrently depends on all its predecessors, in form of $\hat{\boldsymbol{y}} = \phi(\boldsymbol{h}^{[T]})$.

Interestingly, [7] proposed a Encoder-Decoder-Framework for machine translation, which involves both of these variants. First a many-to-one RNN encodes a sentence of the source language into its last hidden state vector, which is interpreted as the representation for the entire sentence. The second one is a many-to-many RNN. It consumes the last hidden state of the encoder as its first hidden state and generates a sentence of the target language. We illustrate this model using a simple example in Fig. 3.

In their work the many-to-one RNN was proven capable of learning a fixed-size representation from the entire input sequence of variable length, which is an appealing characteristics for our data situation as well. In our data, each patient case has a medical history of variable length, and the number of clinical events observed before a therapy prescription varies between 0 and 35. With such an encoder

Figure 3. The Encoder-Decoder-Framework for Machine Translation by [7]. The encoder RNN outputs only its last hidden state when it sees the end-of-sentence symbol. At the same time, the decoder RNN consumes this hidden state as its initial one and generates the first word. The decoder keeps generating words till it generates the end-of-sentence symbol.



RNN we could extract from such sequential input a more abstract and compact vector representing the entire history of the patient up to a specific time step. For the sake of simplicity, we denote such a many-to-one RNN (either GRU or LSTM) using a function ω:

$$\boldsymbol{h}_i^{[t^*]} = \omega(\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{t^*}), \tag{4}$$

where $\boldsymbol{h}_i^{[t^*]}$ is the last hidden state.

In order to also take into account the static features such as patient information and primary tumor, we follow [5] and concatenate the output of the RNN with the latent representation of the static features.

$$\boldsymbol{z}_i^{[t^*]} = (\boldsymbol{h}_i^{[t^*]}, \boldsymbol{q}_i) \quad \text{with } \boldsymbol{q}_i = \psi(\boldsymbol{H}^T \boldsymbol{m}_i), \tag{5}$$

where $\boldsymbol{H}$ is a usual trainable weight matrix and ψ denotes a non-linear activation function. Therefore, the vector $\boldsymbol{z}_i^{[t^*]}$ represents the static patient information as well as the medical history of patient i up to time step t^*. Such a vector functions as an abstract patient profile that represents all relevant clinical information in a latent vector space, where patients with similar background information and medical history could be encouraged to be placed in a specific neighborhood. This very characteristic of the latent vector space is key to the latest success of Deep Learning, in that it facilitates the classification and regression models built on top of it.

4.2. Hierarchical Response Model as Decoder

We attempt to model the therapies in a similar fashion as the physicians' prediction procedure. A physician first has to choose one therapy category, and then to specify for the chosen category its features. We propose a Multinomial Hierarchical Regression (MHR) to model this procedure.

In the first step we model the probability that each of the three therapy categories is chosen at time step t^* for

patient i using a multinomial variable $C_i^{[t^*]}$ with a softmax activation:

$$\mathbb{P}(C_i^{[t^*]} = k \mid \boldsymbol{m}_i, \{\boldsymbol{x}_i^{[t]}\}_{t=1}^{t^*}) = \frac{\exp\left((\boldsymbol{z}_i^{[t^*]})^T \boldsymbol{\gamma}^k\right)}{\sum_{\forall k'} \exp\left((\boldsymbol{z}_i^{[t^*]})^T \boldsymbol{\gamma}^{k'}\right)}, \tag{6}$$

where $\boldsymbol{z}_i^{[t^*]}$, as defined in Eq. (5), is the latent representation for the patient up to this time step and $\boldsymbol{\gamma}^k$ serves as the category-specific parameter vector.

Then in the second step, given a specific therapy category k, we denote the number of therapy features in this category with L^k and model the l^k-th multinomial distributed feature variable F_{k,l^k}, whose conditional probability can be modeled with

$$\begin{aligned} &\mathbb{P}(F_{i,k,l^k}^{[t^*]} = r \mid C_i^{[t^*]} = k, \boldsymbol{m}_i, \{\boldsymbol{x}_i^{[t]}\}_{t=1}^{t^*}) = \\ =&\frac{\exp\left((\boldsymbol{z}_i^{[t^*]})^T \boldsymbol{\beta}_{k,l^k,r}\right)}{\sum_{\forall r'} \exp\left((\boldsymbol{z}_i^{[t^*]})^T \boldsymbol{\beta}_{k,l^k,r'}\right)}, \end{aligned} \tag{7}$$

if k=1 or k=2, i.e. in case of radiotherapy or systemic therapy where therapy features in each category are multiple multinomial distributed. Therefore one would need the softmax function to model the probabilities that the therapy feature takes one specific value r. We denote the parameter vector $\boldsymbol{\beta}_{k,l^k,r}$ with three levels of subscripts: k suggests the category of the therapy, l^k selects one specific multinomial feature from this category, and r denotes the r-th possible outcome of this feature. For instance, we would use $\boldsymbol{\beta}_{1,2,3}$ to denote the parameters corresponding to the hierarchy of `radiotherapy` / `type` / `other_radiotherapy`, implying that the type of the radiotherapy is of other kinds (3rd column, 6th row in Tab. 3).

If the therapy category suggests the surgery, i.e. k=3, whose features consist of L_k=10 Bernoulli variable, we would have instead of Eq. (7) the following formulation:

$$\begin{aligned} &\mathbb{P}(F_{i,k,l^k}^{[t^*]} = r \mid C_i^{[t^*]} = k, \boldsymbol{m}_i, \{\boldsymbol{x}_i^{[t]}\}_{t=1}^{t^*}) \\ =&\sigma\left((\boldsymbol{z}_i^{[t^*]})^T \boldsymbol{\beta}_{k,l^k,r}\right), \end{aligned} \tag{8}$$

with $r = 1$ in all cases, because a Bernoulli variable has an one-dimensional outcome.

The product of Eq. (6) and (7) as well as that of Eq. (6) and (8) yields the joint probability of both therapy feature and category as

$$\mathbb{P}(F_{i,k,l^k}^{[t^*]} = r \wedge C_i^{[t^*]} = k \mid \boldsymbol{m}_i, \{\boldsymbol{x}_i^{[t]}\}_{t=1}^{t^*}). \tag{9}$$

But due to the fact that

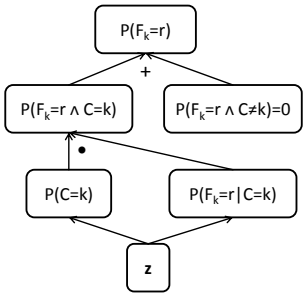
$$\mathbb{P}(F_{i,k,l^k}^{[t^*]} = r \wedge C_i^{[t^*]} \neq k \mid \boldsymbol{m}_i, \{\boldsymbol{x}_i^{[t]}\}_{t=1}^{t^*}) = 0, \tag{10}$$

in all cases, this joint probability of Eq. (9) is equal to

$$\mathbb{P}(F_{i,k,l^k}^{[t^*]} = r | \boldsymbol{m}_i, \{\boldsymbol{x}_i^{[t]}\}_{t=1}^{t^*}), \tag{11}$$

applying the law of total probability, yielding the marginal prediction and allowing us to perform the optimization against the target vector. The calculation with these probabilities is illustrated in Fig. 4. A same design can also be found in [14], where they factorize a large softmax layer into such a tree-like hierarchy.

Figure 4. A simplified illustration of deriving the marginal probability of the therapy feature. From the vector $\boldsymbol{z}$ representing a patient one calculates the category probability and the feature probability conditioned on the category. Then product of the two yields the joint probability of feature and category. Combined with the joint probability of feature and non-category, which is always 0, one would get the marginal probability of the feature.



In [13] a very similar approach is referred to as the Multinomial Model with Hierarchically Structured Response. The major difference lies in the fact that in [13] only one multinomial response on the second level is linked with each category on the first level. This is apparently not sufficient for our data situation where multiple multinomial therapy features fall into each therapy category. Therefore we extend this model and allow for multiple of such links.

Finally we illustrate the complete model architecture in Fig. 5. There the RNN encoder outputs its last hidden state that represents the whole sequence and is concatenated with the latent representation mapped from the static patient information. This concatenated vector forms the input to the hierarchical model, which in the first step calculates the therapy category probabilities and in the second step the therapy feature probabilities conditioned on corresponding category. These two levels of probabilities are multiplied, giving the joint probabilities of category and feature, which are equivalent to marginal feature probabilities as proven in Eq. (10).

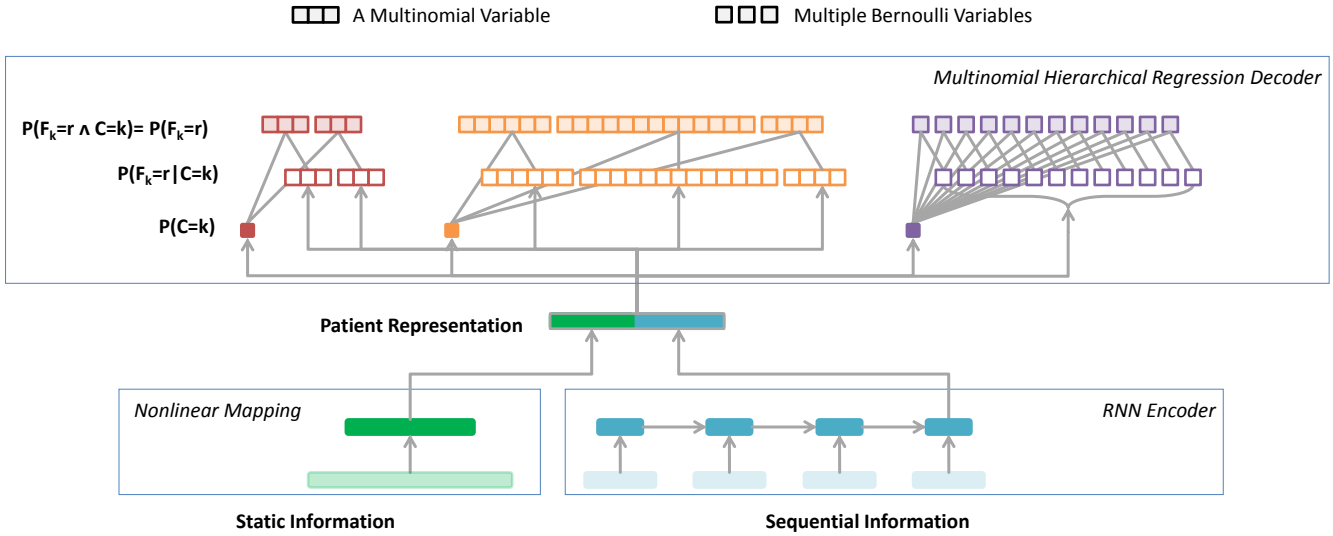
5. Experiment

We evaluate our encoder-decoder model from two aspects. First we assess the prediction quality and then demonstrate that our model can be exploited to identify similar historical patient cases in a very efficient way.

5.1. Modeling of Therapy Decisions

In order to take into account the prediction stability, we conduct cross-validation by splitting the 2,869 patients into 5 disjoint sets, and then query their corresponding sequences to form the training and test sets. In contrast to performing the splitting on the level of sequences, this approach guarantees

Figure 5. Our proposed model architecture. The radiotherapy features consist of two 3D multinomial variables (red-colored). The systemic therapies consist of on 4D, one 16D and one 3D multinomial variables (orange-colored). The surgery feature consists of 10 Bernoulli variables (purple-colored).



that the model only predicts for completely new patients whose information was never –not even partially– present during training, making the experiments more challenging and realistic. For the rest of this section we report the average performances of cross-validations, for all experimental settings including baseline models.

With respect to the sizes of $\boldsymbol{q}_i$ that represents the static information and $\boldsymbol{h}_i^{[t^*]}$ that represents the medical history, we conduct experiments with two settings. In a smaller setting we define $\boldsymbol{q}_i \in \mathbb{R}^{64}$ and $\boldsymbol{h}_i^{[t^*]} \in \mathbb{R}^{128}$ and present the results in Tab. 5, while Tab. 6 provides experimental results with a larger setting of $\boldsymbol{q}_i \in \mathbb{R}^{128}$ and $\boldsymbol{h}_i^{[t^*]} \in \mathbb{R}^{256}$.

Further hyper-parameters are set as follows: the output of $\boldsymbol{h}_i^{[t^*]}$ in RNN is activated with $\tanh$. We apply 0.25 Dropout [23] for weights in RNNs and 0.001 Ridge penalization for the MHR and logistic decoders. Each model instance is trained with Adam [24] step rule for a maximum of 1000 iterations with early stopping mechanism.

We present two classes of evaluation metrics. First, column-wise average Area Under ROC (AUROC) and Area Under Precision-Recall-Curve (AUPRC), which are well-known metrics applied to measure the classification quality, should indicate the models' capability to assign patients to the correct therapy features. Secondly, we report multi-label ranking-based metrics of Coverage Error (CE) [25] and Label Ranking Average Precision (LRAP) [26] in the scikit-learn library [27]. In contrast to precision and recall based metrics, they are calculated row-wise and thus evaluate for each patient how many recommended therapies were actually prescribed. LRAP ranges between 0 and 1 just as AUROC and AUPRC. CE describes how many steps one has to go in a ranked list of recommendations till one covers all ground truth labels. In our case, the average number of labels in each patient case is 4.4 and the total number of possible labels is 39. The CE shall therefore be ideally 4.4, suggesting a perfect prediction, and be 39 in worst case scenario (Tab. 4).

Table 4. Results of experiments with two weak baselines: Random Prediction and Constant Most Popular Prediction.

Weak Baselines	AUROC	AUPRC	CE	LRAP
Random	49.7%	9.4%	38.2	11.2%
Most Popular	50.0%	21.3%	13.9	38.6%

We experiment with three encoders and two decoders.

The baseline encoder is a simple Feed-Forward Layer (FFL) consuming the raw sequential information that is aggregated with respect to time. Then the aggregated feature vector is concatenated with the static feature vector for each patient case. Such aggregation can be interpreted as a hand-engineered feature processing, where each feature represents the total number of observed feature values. It also corresponds to the bag-of-words approach [28] in Natural Language modeling, this approach completely neglects the *order* in which the feature values are observed. As a more advanced solution we apply GRU and LSTM as RNN encoders as introduced in Section 4.1, which are expected to capture the information regarding the events order as well.

The baseline decoder is a single-layered logistic regression, which is a popular choice in multi-class multi-label classification tasks in machine learning. Please note that this approach does not fully satisfy the distribution assumption of the target. For instance, a therapy *feature* variable is multinomially distributed, implying the mutual exclusiveness of the probable outcomes of the feature values and this aspect cannot be taken into account with a flat logistic regression. Such mutual exclusiveness has to be taken into account especially in clinical data. For instance a physician is only supposed to prescribe one medication from a class of related medications. Since our proposed MHR model, presented in

Sec. 4.2, is mathematically solid from this perspective, it is interesting to see it actually outperforms a more popular but less accurate alternative.

We conduct experiments applying all possible combinations of encoders and decoders, to identify i) which combination yields the best prediction performance and ii) which encoder contributes the most to the model performances given the same decoder, and vice versa.

Table 5. AVERAGE RESULTS OF EXPERIMENTS WITH DIFFERENT ENCODERS AND DECODERS, WITH $\boldsymbol{q}_i \in \mathbb{R}^{128}$ AND $\boldsymbol{h}_i^{[t^*]} \in \mathbb{R}^{256}$

Encoder	Decoder	AUROC	AUPRC	CE	LRAP
FFL	Logistic	69.4%	13.4%	12.61	48.6%
	MHR	70.3%	13.9%	11.79	49.3%
GRU	Logistic	81.8%	28.8%	8.57	61.3%
	MHR	**82.1%**	**31.2%**	**8.26**	**62.3%**
LSTM	Logistic	79.6%	24.7%	9.47	57.9%
	MHR	81.9%	30.2%	8.53	61.4%

Table 6. AVERAGE RESULTS OF EXPERIMENTS WITH DIFFERENT ENCODERS AND DECODERS, WITH $\boldsymbol{q}_i \in \mathbb{R}^{64}$ AND $\boldsymbol{h}_i^{[t^*]} \in \mathbb{R}^{128}$

Encoder	Decoder	AUROC	AUPRC	CE	LRAP
FFL	Logistic	69.8%	13.4%	12.83	48.3%
	MHR	70.2%	13.9%	11.83	49.2%
GRU	Logistic	80.0%	26.2%	9.28	59.0%
	MHR	**81.3%**	**28.2%**	**8.71**	**61.3%**
LSTM	Logistic	78.7%	23.0%	9.93	56.4%
	MHR	80.6%	26.7%	9.12	59.5%

Comparing Tab. 5 with 6 one could observe that, with a larger size for the representation vector, the prediction quality can be improved in almost all cases. With both parameter settings the combination of GRU encoder and the hierarchical decoder yields the best quality scores.

It is to note that both decoders on top of the baseline FFL encoder show suboptimal results compared with those on top of RNN encoders, i.e., GRU and LSTM encoders significantly boost the prediction quality even with a mere logistic regression as decoder. On the other hand, the MHR model further improves the prediction quality in comparison with a flat logistic regression. This suggests that the RNN encoders contribute a larger proportion to the prediction quality, while the multinomial hierarchical decoder alone does not improve the model to a significant extent without a decent encoder model. One could draw the conclusion that the encoded representation of a patient case plays a central role in this model. In total, the prediction best quality is provided by GRU encoder and MHR decoder.

5.2. Identification of Similar Patient Cases

In a realistic application scenario in a clinic, it is as important to provide physicians with recommended therapies as to provide a list of similar patient cases. If the set of similar patient cases have received therapies similar to the recommended ones, it will support these recommendations and encourage the physician to interpret the recommendations with more confidence. But due to the fact that patients have medical histories of variable lengths, it is nontrivial to apply common distance metrics directly on the patient features to quantify the similarity. For instance, it is impossible to mathematically directly calculate the distance between a patient having undergone a breast conservation surgery and another patient with one mastectomy followed by three successive radiotherapies, although it might be obvious for a physician to tell the difference/similarity.

To this end, we propose that the derived latent vector of $\boldsymbol{z}_i^{[t^*]}$, representing the patient i's profile up to time t^*, can be exploited to identify similar patient cases, since all such vectors have the same dimension.

Using a trained encoder network, we map all training patient cases and the a test case into the latent feature space and define there a k-NN model. The k training cases neighboring the test case can therefore provide a prediction for the test case. This approach reusing trained representation is closely related to the so-called transfer learning [29], where one exploits the latent representations learned for one task for new tasks. In our case, however, we have the same task solved by a new predictive model that consumes the learned representations.

If the k-NN model is able to identify patient cases having received therapies that agree with the recommended ones, then the latent vectors correctly represent such similarity.

We report the results of such k-NN models that is applied on latent representations originally learned with GRU and LSTM encoder combined with logistic and hierarchical decoder in Tab. 7 and 8 for the two parameter settings of the latent vector sizes, respectively. Please note that an RNN encoder converges to different weight parameters combined with different decoders. Therefore, although we only apply the encoder network, it is necessary to differentiate between the two decoder cases. The k is here set to be 30. We realize that a smaller k would hurt the prediction quality and a larger k does not further improve the model.

Table 7. RESULTS OF EXPERIMENTS WITH k-NN ON TOP OF THE LATENT REPRESENTATIONS DERIVED BY DIFFERENT MODEL ARCHITECTURE SETTINGS, WITH $\boldsymbol{q}_i \in \mathbb{R}^{128}$ AND $\boldsymbol{h}_i^{[t^*]} \in \mathbb{R}^{256}$.

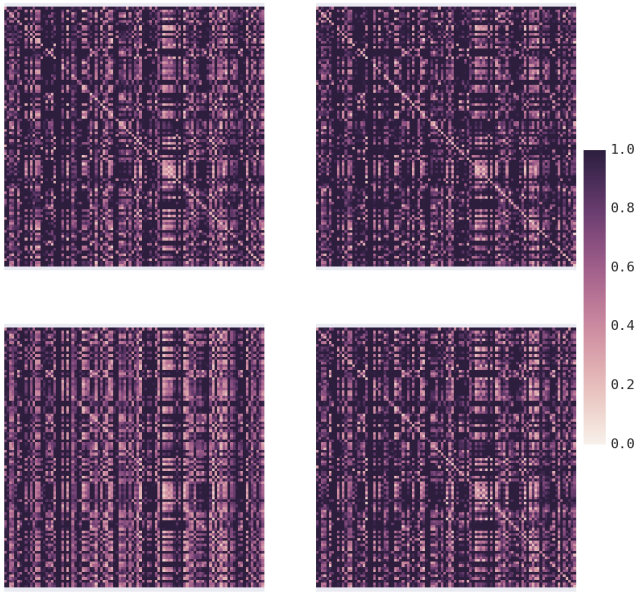
orig. Encoder	orig.Decoder	AUROC	AUPRC	CE	LRAP
GRU	Logistic	78.7%	30.0%	**9.69**	63.0%
	MHR	**79.3%**	**32.2%**	9.82	63.2%
LSTM	Logistic	78.6%	28.7%	9.80	62.7%
	MHR	79.0%	**32.2%**	9.83	**63.3** %

Table 8. RESULTS OF EXPERIMENTS WITH kNN ON TOP OF THE LATENT REPRESENTATIONS DERIVED BY DIFFERENT MODEL ARCHITECTURE SETTINGS, WITH $\boldsymbol{q}_i \in \mathbb{R}^{64}$ AND $\boldsymbol{h}_i^{[t^*]} \in \mathbb{R}^{128}$

orig. Encoder	orig.Decoder	AUROC	AUPRC	CE	LRAP
GRU	Logistic	78.1%	28.6%	9.75	62.9%
	MHR	**79.1%**	29.8%	9.71	**63.3%**
LSTM	Logistic	78.3%	28.5%	9.78	62.5%
	MHR	**79.1%**	**30.2%**	**9.69**	63.2%

One could observe that the k-NN performances are in total quite close to those reported in Tab. 5 and 6.

Figure 6. The distance matrix between a sample of 100 decisions predicted by i) the encoder-decoder model and ii) the k-NN model based on the latent representations with four different encoder-decoder settings. Top-left: GRU+sigmoid; top-right: GRU+hierarchical; bottom-left: LSTM+sigmoid; bottom-right: LSTM+hierarchical.



However, if we build the same k-NN classifier on top of the raw features aggregated in time and concatenated with the static features, the prediction quality is observed to be much worse: The AUROC and AUPRC decrease to 75.2% and 23.3%, respectively, while the CE and LRAP to 11.61 and 56.8%, respectively. This suggests that the RNN encoder is capable of generating more dense and informative latent features for each patient case.

In order to compare the concrete decision made by the original encoder-decoder model and the k-NN model on each specific patient case, we calculate the Euclidean distances between the predictions made by i) the encoder-decoder model and ii) the k-NN model based on the same encoder and decoder setting for each patient case. We visualize in Fig. 6 the distance matrices as heat map. One could observe that the diagonal entries, which represent prediction distances between the complete model and the k-NN model for the same patient cases, are systematically lower in value.

To this end, we argue that the latent vectors can represent a patient case with medical history of variable length in a unified vector space, where the topological characteristics of patients are well preserved. This additional feature of our encoder-decoder model enables the identification of similar patient cases, which can support and supplement the predictive recommendations.

6. Conclusion

We have proposed an Encoder-Decoder network that predicts physicians' therapy decisions as well as provides a list of similar patient cases. The model consists of an RNN encoder that learns an abstract representation of the patient profile, and a hierarchical regression that decodes the latent representation into therapy predictions. Such a predictive model can serve to support clinical decisions, to detect anomalous prescriptions and to support physician by searching for similar historical cases.

We have conducted experiments on a large real-world dataset collected from almost three thousands of metastatic breast cancer patients. The experimental results demonstrate that the RNN encoder greatly improves the modeling quality compared with plain feed-forward models that consume aggregated sequential features. The hierarchical regression model also outperforms a flat logistic regression as a decoder. We have also shown that our model is capable of providing lists of similar patient cases, although it is nontrivial to measure distance among patients, when they all have medical histories of variable lengths.

The generic contribution of this work consists of following aspects:

- We transfer the popular Encoder-Decoder architecture from NLP to the clinical domain;
- We propose a hierarchical classifier that mimics the actual multi-step decision procedure;
- We empirically prove that the latent vector representing each patient case produced by RNN encoders in general facilitates the prediction with k-NN, logistic regression and MHR;
- We showed that such latent representations can be exploited to identify similar patients with higher quality than with aggregated sequential features.

Encouraged by the success of the RNN models in handling sequential data, one interesting and realistic improvement of the model would be to integrate attention mechanisms [30, 31] into the RNN encoder. The model would, for instance, be able to identify which historical event has contributed most to the decision, which could further improve the model's interpretability and encourage its application in clinics.

References

[1] V. Tresp, M. Overhage, M. Bundschus, S. Rabizadeh, P. Fasching, and S. Yu, "Going digital: A survey on digitalization and large scale data analytics in healthcare," *arXiv preprint arXiv:1606.08075*, 2016.

[2] R. Rahman and C. K. Reddy, "Electronic health records: a survey," *Healthcare Data Analytics*, vol. 36, p. 21, 2015.

[3] E. Choi, M. T. Bahadori, and J. Sun, "Doctor ai: Predicting clinical events via recurrent neural networks," *arXiv preprint arXiv:1511.05942*, 2015.

[4] C. Esteban, D. Schmidt, D. Krompaß, and V. Tresp, "Predicting sequences of clinical events by using a personalized temporal latent embedding model," in *Healthcare Informatics (ICHI), 2015 International Conference on*. IEEE, 2015, pp. 130–139.

[5] C. Esteban, O. Staeck, Y. Yang, and V. Tresp, "Predicting clinical events by combining static and dynamic information using recurrent neural networks," *arXiv preprint arXiv:1602.02685*, 2016.

[6] M. A. Brookhart, T. Stürmer, R. J. Glynn, J. Rassen, and S. Schneeweiss, "Confounding control in healthcare database research: challenges and potential approaches," *Medical care*, vol. 48, no. 6 0, p. S114, 2010.

[7] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[8] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[9] Z. C. Lipton, D. C. Kale, C. Elkan, and R. Wetzell, "Learning to diagnose with lstm recurrent neural networks," *arXiv preprint arXiv:1511.03677*, 2015.

[10] E. Choi, M. T. Bahadori, J. Sun, J. Kulas, A. Schuetz, and W. Stewart, "Retain: An interpretable predictive model for healthcare using reverse time attention mechanism," in *Advances in Neural Information Processing Systems*, 2016, pp. 3504–3512.

[11] J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber, "A clockwork rnn," *arXiv preprint arXiv:1402.3511*, 2014.

[12] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 2011, pp. 142–150.

[13] G. Tutz, *Regression for Categorical Data:*, ser. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2011.

[14] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model." in *Aistats*, vol. 5. Citeseer, 2005, pp. 246–252.

[15] P. Kaatsch, C. Spix, S. Hentschel, A. Katalinic, S. Luttmann, C. Stegmaier, S. Caspritz, J. Cernaj, A. Ernst, J. Folkerts *et al.*, "Krebs in deutschland 2009/2010," 2013.

[16] C. Rauh and W. Matthias, "Interdisziplinäre s3-leitlinie für die diagnostik, therapie und nachsorge des mammakarzinoms," 2008.

[17] P. Fasching, S. Brucker, T. Fehm, F. Overkamp, W. Janni, M. Wallwiener, P. Hadji, E. Belleville, L. Häberle, F. Taran, D. Luftner, M. Lux, J. Ettl, V. Muller, H. Tesch, D. Wallwiener, and A. Schneeweiss, "Biomarkers in patients with metastatic breast cancer and the praegnant study network," *Geburtshilfe Frauenheilkunde*, vol. 75, no. 01, pp. 41–50, 2015. [Online]. Available: http://www.praegnant.org/

[18] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[21] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-aware neural language models," *arXiv preprint arXiv:1508.06615*, 2015.

[22] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Interspeech*, vol. 2, 2010, p. 3.

[23] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[24] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[25] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining multi-label data," in *Data mining and knowledge discovery handbook*. Springer, 2009, pp. 667–685.

[26] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Džeroski, "An extensive experimental comparison of methods for multi-label learning," *Pattern Recognition*, vol. 45, no. 9, pp. 3084–3104, 2012.

[27] 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, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[28] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[29] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[30] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Advances in Neural Information Processing Systems*, 2015, pp. 577–585.

[31] V. Mnih, N. Heess, A. Graves *et al.*, "Recurrent models of visual attention," in *Advances in neural information processing systems*, 2014, pp. 2204–2212.

Chapter 4

Tensor-Train RNNs for Video Classification

Tensor-Train Recurrent Neural Networks for Video Classification

Yinchong Yang [1 2] **Denis Krompass** [2] **Volker Tresp** [1 2]

Abstract

The Recurrent Neural Networks and their variants have shown promising performances in sequence modeling tasks such as Natural Language Processing. These models, however, turn out to be impractical and difficult to train when exposed to very high-dimensional inputs due to the large input-to-hidden weight matrix. This may have prevented RNNs' large-scale application in tasks that involve very high input dimensions such as video modeling; current approaches reduce the input dimensions using various feature extractors. To address this challenge, we propose a new, more general and efficient approach by factorizing the input-to-hidden weight matrix using Tensor-Train decomposition which is trained simultaneously with the weights themselves. We test our model on classification tasks using multiple real-world video datasets and achieve competitive performances with state-of-the-art models, even though our model architecture is orders of magnitude less complex. We believe that the proposed approach provides a novel and fundamental building block for modeling high-dimensional sequential data with RNN architectures and opens up many possibilities to transfer the expressive and advanced architectures from other domains such as NLP to modeling high-dimensional sequential data.

1. Introduction

Nowadays, the Recurrent Neural Network (RNN), especially its more advanced variants such as the LSTM and the GRU, belong to the most successful machine learning approaches when it comes to sequence modeling. Especially in Natural Language Processing (NLP), great improvements have been achieved by exploiting these Neural Network architectures. This success motivates efforts to also apply these RNNs to video data, since a video clip could be seen as a sequence of image frames. However, plain RNN models turn out to be impractical and difficult to train directly on video data due to the fact that each image frame typically forms a relatively high-dimensional input, which makes the weight matrix mapping from the input to the hidden layer in RNNs extremely large. For instance, in case of an RGB video clip with a frame size of say $160\times120\times3$, the input vector for the RNN would already be $57,600$ at each time step. In this case, even a small hidden layer consisting of only 100 hidden nodes would lead to 5,760,000 free parameters, only considering the input-to-hidden mapping in the model.

In order to circumvent this problem, state-of-the-art approaches often involve pre-processing each frame using Convolution Neural Networks (CNN), a Neural Network model proven to be most successful in image modeling. The CNNs do not only reduce the input dimension, but can also generate more compact and informative representations that serve as input to the RNN. Intuitive and tempting as it is, training such a model from scratch in an end-to-end fashion turns out to be impractical for large video datasets. Thus, many current works following this concept focus on the CNN part and reduce the size of RNN in term of sequence length (Donahue et al., 2015; Srivastava et al., 2015), while other works exploit pre-trained deep CNNs as pre-processor to generate static features as input to RNNs (Yue-Hei Ng et al., 2015; Donahue et al., 2015; Sharma et al., 2015). The former approach neglects the capability of RNNs to handle sequences of variable lengths and therefore does not scale to larger, more realistic video data. The second approach might suffer from suboptimal weight parameters by not being trained end-to-end (Fernando & Gould, 2016). Furthermore, since these CNNs are pre-trained on existing image datasets, it remains unclear how well the CNNs can generalize to video frames that could be of totally different nature from the image training sets.

Alternative approaches were earlier applied to generate image representations using dimension reductions such as PCA (Zhang et al., 1997; Kambhatla & Leen, 1997; Ye et al., 2004) and Random Projection (Bingham & Mannila, 2001). Classifiers were built on such features to perform object and face recognition tasks. These models, however,

[1]Ludwig Maximilian University of Munich, Germany [2]Siemens AG, Corporate Technology, Germany. Correspondence to: Yinchong Yang <yinchong.yang@siemens.com>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

are often restricted to be linear and cannot be trained jointly with the classifier.

In this work, we pursue a new direction where the RNN is exposed to the raw pixels on each frame without any CNN being involved. At each time step, the RNN first maps the large pixel input to a latent vector in a typically much lower dimensional space. Recurrently, each latent vector is then enriched by its predecessor at the last time step with a hidden-to-hidden mapping. In this way, the RNN is expected to capture the inter-frame transition patterns to extract the representation for the entire sequence of frames, analogous to RNNs generating a sentence representation based on word embeddings in NLP (Sutskever et al., 2014). In comparison with other mapping techniques, a direct input-to-hidden mapping in an RNN has several advantages. First it is much simpler to train than deep CNNs in an end-to-end fashion. Secondly it is exposed to the complete pixel input without the linear limitation as PCA and Random Projection. Thirdly and most importantly, since the input-to-hidden and hidden-to-hidden mappings are trained jointly, the RNN is expected to capture the correlation between spatial and temporal patterns.

To address the issue of having too large of a weight matrix for the input-to-hidden mapping in RNN models, we propose to factorize the matrix with the Tensor-Train decomposition (Oseledets, 2011). In (Novikov et al., 2015) the Tensor-Train has been applied to factorize a fully-connected feed-forward layer that can consume image pixels as well as latent features. We conducted experiments on three large-scale video datasets that are popular benchmarks in the community, and give empirical proof that the proposed approach makes very simple RNN architectures competitive with the state-of-the-art models, even though they are of several orders of magnitude lower complexity.

The rest of the paper is organized as follows: In Section 2 we summarize the state-of-the-art works, especially in video classification using Neural Network models and the tensorization of weight matrices. In Section 3 we first introduce the Tensor-Train model and then provide a detailed derivation of our proposed Tensor-Train RNNs. In Section 4 we present our experimental results on three large scale video datasets. Finally, Section 5 serves as a wrap-up of our current contribution and provides an outlook of future work.

Notation We index an entry in a d-dimensional tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{p_1 \times p_2 \times \ldots \times p_d}$ using round parentheses such as $\boldsymbol{\mathcal{A}}(l_1, l_2, ..., l_d) \in \mathbb{R}$ and $\boldsymbol{\mathcal{A}}(l_1) \in \mathbb{R}^{p_2 \times p_3 \times \ldots \times p_d}$, when we only write the first index. Similarly, we also use $\boldsymbol{\mathcal{A}}(l_1, l_2) \in \mathbb{R}^{p_3 \times p_4 \times \ldots \times p_d}$ to refer to the sub-tensor specified by two indices l_1 and l_2.

2. Related Works

The current approaches to model video data are closely related to models for image data. A large majority of these works use deep CNNs to process each frame as image, and aggregate the CNN outputs. (Karpathy et al., 2014) proposes multiple fusion techniques such as Early, Late and Slow Fusions, covering different aspects of the video. This approach, however, does not fully take the order of frames into account. (Yue-Hei Ng et al., 2015) and (Fernando & Gould, 2016) apply global pooling of frame-wise CNNs, before feeding the aggregated information to the final classifier. An intuitive and appealing idea is to fuse these frame-wise spatial representations learned by CNNs using RNNs. The major challenge, however, is the computation complexity; and for this reason multiple compromises in the model design have to be made: (Srivastava et al., 2015) restricts the length of the sequences to be 16, while (Sharma et al., 2015) and (Donahue et al., 2015) use pre-trained CNNs. (Xingjian et al., 2015) proposed a more compact solution that applies convolutional layers as input-to-hidden and hidden-to-hidden mapping in LSTM. However, they did not show its performance on large-scale video data. (Simonyan & Zisserman, 2014) applied two stacked CNNs, one for spatial features and the other for temporal ones, and fused the outcomes of both using averaging and a Support-Vector Machine as classifier. This approach is further enhanced with Residual Networks in (Feichtenhofer et al., 2016). To the best of our knowledge, there has been no published work on applying pure RNN models to video classification or related tasks.

The Tensor-Train was first introduced by (Oseledets, 2011) as a tensor factorization model with the advantage of being capable of scaling to an arbitrary number of dimensions. (Novikov et al., 2015) showed that one could reshape a fully connected layer into a high-dimensional tensor and then factorize this tensor using Tensor-Train. This was applied to compress very large weight matrices in deep Neural Networks where the entire model was trained end-to-end. In these experiments they compressed fully connected layers on top of convolution layers, and also proved that a Tensor-Train Layer can directly consume pixels of image data such as CIFAR-10, achieving the best result among all known non-convolutional models. Then in (Garipov et al., 2016) it was shown that even the convolutional layers themselves can be compressed with Tensor-Train Layers. Actually, in an earlier work by (Lebedev et al., 2014) a similar approach had also been introduced, but their CP factorization is calculated in a pre-processing step and is only fine tuned with error back propagation as a post processing step.

(Koutnik et al., 2014) performed two sequence classification tasks using multiple RNN architectures of relatively low dimensionality: The first task was to classify spoken

words where the input sequence had a dimension of 13 channels. In the second task, RNNs were trained to classify handwriting based on the time-stamped 4D spatial features. RNNs have been also applied to classify the sentiment of a sentence such as in the IMDB reviews dataset (Maas et al., 2011). In this case, the word embeddings form the input to RNN models and they may have a dimension of a few hundreds. The sequence classification model can be seen as a special case of the Encoder-Decoder-Framework (Sutskever et al., 2014) in the sense that a classifier decodes the learned representation for the entire sequence into a probabilistic distribution over all classes.

3. Tensor-Train RNN

In this section, we first give an introduction to the core ingredient of our proposed approach, i.e., the Tensor-Train Factorization, and then use this to formulate a so-called Tensor-Train Layer (Novikov et al., 2015) which replaces the weight matrix mapping from the input vector to the hidden layer in RNN models. We emphasize that such a layer is learned end-to-end, together with the rest of the RNN in a very efficient way.

3.1. Tensor-Train Factorization

A *Tensor-Train Factorization* (TTF) is a tensor factorization model that can scale to an arbitrary number of dimensions. Assuming a d-dimensional target tensor of the form $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{p_1 \times p_2 \times ... \times p_d}$, it can be factorized in form of:

$$\widehat{\boldsymbol{\mathcal{A}}}(l_1, l_2, ..., l_d) \overset{TTF}{=} \boldsymbol{\mathcal{G}}_1(l_1)\, \boldsymbol{\mathcal{G}}_2(l_2)\, ...\, \boldsymbol{\mathcal{G}}_d(l_d) \tag{1}$$

where

$$\begin{aligned} &\boldsymbol{\mathcal{G}}_k \in \mathbb{R}^{p_k \times r_{k-1} \times r_k},\ l_k \in [1, p_k]\ \forall k \in [1, d] \\ &\text{and}\ \ r_0 = r_d = 1. \end{aligned} \tag{2}$$

As Eq. 1 suggests, each entry in the target tensor is represented as a sequence of matrix multiplications. The set of tensors $\{\boldsymbol{\mathcal{G}}_k\}_{k=1}^d$ are usually called core-tensors. The complexity of the TTF is determined by the ranks $[r_0, r_1, ..., r_d]$. We demonstrate this calculation also in Fig. 1. Please note that the dimensions and core-tensors are indexed from 1 to d while the rank index starts from 0; also note that the first and last ranks are both restricted to be 1, which implies that the first and last core tensors can be seen as matrices so that the outcome of the chain of multiplications in Eq. 1 is always a scalar.

If one imposes the constraint that each integer p_k as in Eq. (1) can be factorized as $p_k = m_k \cdot n_k\ \forall k \in [1, d]$, and consequently reshapes each $\boldsymbol{\mathcal{G}}_k$ into $\boldsymbol{\mathcal{G}}_k^* \in \mathbb{R}^{m_k \times n_k \times r_{k-1} \times r_k}$, then each index l_k in Eq. (1) and (2) can be uniquely represented with two indices (i_k, j_k), i.e.

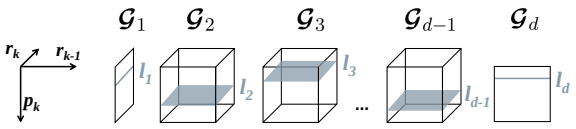


Figure 1: Tensor-Train Factorization Model: To reconstruct one entry in the target tensor, one performs a sequence of vector-matrix-vector multiplications, yielding a scalar.

$$i_k = \lfloor \frac{l_k}{n_k} \rfloor,\ j_k = l_k - n_k \lfloor \frac{l_k}{n_k} \rfloor, \tag{3}$$

$$\text{so that}\ \ \boldsymbol{\mathcal{G}}_k(l_k) = \boldsymbol{\mathcal{G}}_k^*(i_k, j_k) \in \mathbb{R}^{r_{k-1} \times r_k}. \tag{4}$$

Correspondingly, the factorization for the tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{(m_1 \cdot n_1) \times (m_2 \cdot n_2) \times ... \times (m_d \cdot n_d)}$ can be rewritten equivalently to Eq.(1):

$$\begin{aligned} &\widehat{\boldsymbol{\mathcal{A}}}((i_1, j_1), (i_2, j_2), ..., (i_d, j_d)) \\ &\overset{TTF}{=} \boldsymbol{\mathcal{G}}_1^*(i_1, j_1)\, \boldsymbol{\mathcal{G}}_2^*(i_2, j_2)\, ...\, \boldsymbol{\mathcal{G}}_d^*(i_d, j_d). \end{aligned} \tag{5}$$

This double index trick (Novikov et al., 2015) enables the factorizing of weight matrices in a feed-forward layer as described next.

3.2. Tensor-Train Factorization of a Feed-Forward Layer

Here we factorize the weight matrix $\boldsymbol{W}$ of a fully-connected feed-forward layer denoted in $\hat{\boldsymbol{y}} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$.

First we rewrite this layer in an equivalent way with scalars as:

$$\begin{aligned} &\hat{\boldsymbol{y}}(j) = \sum_{i=1}^{M} \boldsymbol{W}(i, j) \cdot \boldsymbol{x}(i) + \boldsymbol{b}(j) \\ &\forall j \in [1, N] \text{ and with } \ \boldsymbol{x} \in \mathbb{R}^M,\ \boldsymbol{y} \in \mathbb{R}^N. \end{aligned} \tag{6}$$

Then, if we assume that $M = \prod_{k=1}^d m_k$, $N = \prod_{k=1}^d n_k$ i.e. both M and N can be factorized into two integer arrays of the same length, then we can reshape the input vector $\boldsymbol{x}$ and the output vector $\hat{\boldsymbol{y}}$ into two tensors with the same number of dimensions: $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m_1 \times m_2 \times ... \times m_d}, \boldsymbol{\mathcal{Y}} \in \mathbb{R}^{n_1 \times n_2 \times ... \times n_d}$, and the mapping function $\mathbb{R}^{m_1 \times m_2 \times ... \times m_d} \to \mathbb{R}^{n_1 \times n_2 \times ... \times n_d}$ can be written as:

$$\begin{aligned} &\widehat{\boldsymbol{\mathcal{Y}}}(j_1, j_2, ..., j_d) \\ &= \sum_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} ... \sum_{i_d=1}^{m_d} \boldsymbol{\mathcal{W}}((i_1, j_1), (i_2, j_2), ..., (i_d, j_d)) \cdot \\ &\quad \boldsymbol{\mathcal{X}}(i_1, i_2, ..., i_d) + \boldsymbol{\mathcal{B}}(j_1, j_2, ..., j_d). \end{aligned} \tag{7}$$

Note that Eq. (6) can be seen as a special case of Eq. (7) with $d = 1$. The d-dimensional double-indexed tensor of weights $\mathcal{W}$ in Eq.(7) can be replaced by its TTF representation:

$$\begin{aligned}&\widehat{\mathcal{W}}((i_1, j_1), (i_2, j_2), ..., (i_d, j_d))\\ &\overset{TTF}{=} \mathcal{G}_1^*(i_1, j_1)\, \mathcal{G}_2^*(i_2, j_2)\, ...\, \mathcal{G}_d^*(i_d, j_d).\end{aligned} \tag{8}$$

Now instead of explicitly storing the full tensor $\mathcal{W}$ of size $\prod_{k=1}^d m_k \cdot n_k = M \cdot N$, we only store its TT-format, i.e., the set of low-rank core tensors $\{\mathcal{G}_k\}_{k=1}^d$ of size $\sum_{k=1}^d m_k \cdot n_k \cdot r_{k-1} \cdot r_k$, which can approximately reconstruct $\mathcal{W}$.

The forward pass complexity (Novikov et al., 2015) for one scalar in the output vector indexed by $(j_1, j_2, ..., j_d)$ turns out to be $\mathcal{O}(d \cdot \tilde{m} \cdot \tilde{r}^2)$. Since one needs an iteration through all such tuples, yielding $\mathcal{O}(\tilde{n}^d)$, the total complexity for one Feed-Forward-Pass can be expressed as $\mathcal{O}(d \cdot \tilde{m} \cdot \tilde{r}^2 \cdot \tilde{n}^d)$, where $\tilde{m} = \max_{k \in [1,d]} m_k, \tilde{n} = \max_{k \in [1,d]} n_k, \tilde{r} = \max_{k \in [1,d]} r_k$. This, however, would be $\mathcal{O}(M \cdot N)$ for a fully-connected layer.

One could also compute the compression rate as the ratio between the number of weights in a fully connected layer and that in its compressed form as:

$$r = \frac{\sum_{k=1}^d m_k n_k r_{k-1} r_k}{\prod_{k=1}^d m_k n_k}. \tag{9}$$

For instance, an RGB frame of size $160 \times 120 \times 3$ implies an input vector of length 57,600. With a hidden layer of size, say, 256 one would need a weight matrix consisting of 14,745,600 free parameters. On the other hand, a TTL that factorizes the input dimension with $8 \times 20 \times 20 \times 18$ is able to represent this matrix using 2,976 parameters with a TT-rank of 4, or 4,520 parameters with a TT-rank of 5 (Tab. 1), yielding compression rates of 2.0e-4 and 3.1e-4, respectively.

For the rest of the paper, we term a fully-connected layer in form of $\hat{\boldsymbol{y}} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, whose weight matrix $\boldsymbol{W}$ is factorized with TTF, a *Tensor-Train Layer* (TTL) and use the notation

$$\hat{\boldsymbol{y}} = TTL(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{x}), \text{or } TTL(\boldsymbol{W}, \boldsymbol{x}) \tag{10}$$

where in the second case no bias is required. Please also note that, in contrast to (Lebedev et al., 2014) where the weight tensor is firstly factorized using non-linear Least-Square method and then fine-tuned with Back-Propagation, the TTL is always trained end-to-end. For details on the gradients calculations please refer to Section 5 in (Novikov et al., 2015).

3.3. Tensor-Train RNN

In this work we investigate the challenge of modeling high-dimensional sequential data with RNNs. For this reason, we factorize the matrix mapping from the input to the hidden layer with a TTL. For an Simple RNN (SRNN), which is also known as the Elman Network, this mapping is realized as a vector-matrix multiplication, whilst in case of LSTM and GRU, we consider the matrices that map from the input vector to the gating units:

TT-GRU:

$$\begin{aligned}\boldsymbol{r}^{[t]} &= \sigma(TTL(\boldsymbol{W}^r, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^r \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^r)\\ \boldsymbol{z}^{[t]} &= \sigma(TTL(\boldsymbol{W}^z, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^z \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^z)\\ \boldsymbol{d}^{[t]} &= \tanh(TTL(\boldsymbol{W}^d, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^d(\boldsymbol{r}^{[t]} \circ \boldsymbol{h}^{[t-1]}))\\ \boldsymbol{h}^{[t]} &= (1 - \boldsymbol{z}^{[t]}) \circ \boldsymbol{h}^{[t-1]} + \boldsymbol{z}^{[t]} \circ \boldsymbol{d}^{[t]},\end{aligned} \tag{11}$$

TT-LSTM:

$$\begin{aligned}\boldsymbol{k}^{[t]} &= \sigma(TTL(\boldsymbol{W}^k, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^k \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^k)\\ \boldsymbol{f}^{[t]} &= \sigma(TTL(\boldsymbol{W}^f, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^f \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^f)\\ \boldsymbol{o}^{[t]} &= \sigma(TTL(\boldsymbol{W}^o, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^o \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^o)\\ \boldsymbol{g}^{[t]} &= \tanh(TTL(\boldsymbol{W}^g, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^g \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^g)\\ \boldsymbol{c}^{[t]} &= \boldsymbol{f}^{[t]} \circ \boldsymbol{c}^{[t-1]} + \boldsymbol{k}^{[t]} \circ \boldsymbol{g}^{[t]}\\ \boldsymbol{h}^{[t]} &= \boldsymbol{o}^{[t]} \circ \tanh(\boldsymbol{c}^{[t]}).\end{aligned} \tag{12}$$

One can see that LSTM and GRU require 4 and 3 TTLs, respectively, one for each of the gating units. Instead of calculating these TTLs successively (which we call vanilla TT-LSTM and vanilla TT-GRU), we increase n_1 —the first [1] of the factors that form the output size $N = \prod_{k=1}^d n_k$ in a TTL— by a factor of 4 or 3, and concatenate all the gates as one output tensor, thus parallelizing the computation. This trick, inspired by the implementation of standard LSTM and GRU in (Chollet, 2015), can further reduce the number of parameters, where the concatenation is actually participating in the tensorization. The compression rate for the input-to-hidden weight matrix $\boldsymbol{W}$ now becomes

$$r^* = \frac{\sum_{k=1}^d m_k n_k r_{k-1} r_k + (c-1)(m_1 n_1 r_0 r_1)}{c \cdot \prod_{k=1}^d m_k n_k} \tag{13}$$

where $c = 4$ in case of LSTM and 3 in case of GRU,

and one can show that r^* is always smaller than r as in Eq. 9. For the former numerical example of a input frame size $160 \times 120 \times 3$, a vanilla TT-LSTM would simply require 4 times as many parameters as a TTL, which would be 11,904 for rank 4 and 18,080 for rank 5. Applying this trick would, however, yield only 3,360 and 5,000 parameters for both ranks, respectively. We cover other possible settings of this numerical example in Tab. 1.

Finally to construct the classification model, we denote the i-th sequence of variable length T_i as a set of vectors

[1] Though in theory one could of course choose any n_k.

Table 1: A numerical example of compressing with TT-RNNs. Assuming that an input dimension of $160\times120\times3$ is factorized as $8 \times 20 \times 20 \times 18$ and the hidden layer as $4 \times 4 \times 4 \times 4 = 256$, depending on the TT-ranks we calculate the number of parameters necessary for a Fully-Connected (FC) layer, a TTL which is equivalent to TT-SRNN , TT-LSTM and TT-GRU in their respective vanilla and parallelized form. For comparison, typical CNNs for preprocessing images such as AlexNet (Krizhevsky et al., 2012; Han et al., 2015) or GoogLeNet (Szegedy et al., 2015) consist of over 61 and 6 million parameters, respectively.

FC	TT-ranks	TTL	vanilla TT-LSTM	TT-LSTM	vanilla TT-GRU	TT-GRU
14,745,600	3	1,752	7,008	2,040	5,256	1,944
	4	2,976	11,904	3,360	8,928	3,232
	5	4,520	18,080	5,000	13,560	4,840

$\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i}$ with $\boldsymbol{x}_i^{[t]} \in \mathbb{R}^M \forall t$. For video data each $\boldsymbol{x}_i^{[t]}$ would be an RGB frame of 3 dimensions. For the sake of simplicity we denote an RNN model, either with or without TTL, with a function $f(\cdot)$:

$$\boldsymbol{h}_i^{[T_i]} = f(\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i}), \text{ where } \boldsymbol{h}_i^{[T_i]} \in \mathbb{R}^N, \tag{14}$$

which outputs the last hidden layer vector $\boldsymbol{h}_i^{[T_i]}$ out of a sequential input of variable length. This vector can be interpreted as a latent representation of the whole sequence, on top of which a parameterized classifier $\phi(\cdot)$ with either softmax or logistic activation produces the distribution over all J classes:

$$\begin{aligned}\mathbb{P}(\boldsymbol{y}_i = \mathbf{1}|\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i}) &= \phi(\boldsymbol{h}_i^{[T_i]}) \\ &= \phi(f(\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i})) \in [0,1]^J,\end{aligned} \tag{15}$$

The model is also illustrated in Fig. 2:

4. Experiments

In the following, we present our experiments conducted on three large video datasets. These empirical results demonstrate that the integration of the Tensor-Train Layer in plain RNN architectures such as a tensorized LSTM or GRU boosts the classification quality of these models tremendously when directly exposed to high-dimensional input data, such as video data. In addition, even though the plain architectures are of very simple nature and very low complexity opposed to the state-of-the-art solutions on these datasets, it turns out that the integration of the Tensor-Train Layer alone makes these simple networks very competitive to the state-of-the-art, reaching second best results in all cases.

UCF11 Data (Liu et al., 2009)

We first conduct experiments on the UCF11 – earlier known as the YouTube Action Dataset. It contains in total 1600 video clips belonging to 11 classes that summarize the human action visible in each video clip such as basketball shooting, biking, diving etc.. These videos originate from YouTube and have natural background ('in the wild') and a resolution of 320×240. We generate a sequence of RGB frames of size 160×120 from each clip at an fps(frame per second) of 24, corresponding to the standard value in film and television production. The lengths of frame sequences vary therefore between 204 to 1492 with an average of 483.7.

Figure 3: Two samples of frame sequences from the UCF11 dataset. The two rows belong to the classes of basketball shooting and volleyball spiking, respectively.

For both the TT-GRUs and TT-LSTMs the input dimension at each time step is $160 \times 120 \times 3 = 57600$ which is factorized as $8 \times 20 \times 20 \times 18$, the hidden layer is chosen to be $4 \times 4 \times 4 \times 4 = 256$ and the Tensor-Train ranks are $[1, 4, 4, 4, 1]$. A fully-connected layer for such a mapping would have required 14,745,600 parameters to learn, while the input-to-hidden layer in TT-GRU and TT-LSTM consist of only 3,360 and 3,232, respectively.

As the first baseline model we sample 6 random frames in ascending order. The model is a simple Multilayer Perceptron (MLP) with two layers of weight matrices, the first of which being a TTL. The input is the concatenation of all 6 flattened frames and the hidden layer is of the same size as the hidden layer in TT-RNNs. We term this model as Tensor-Train Multilayer Perceptron (TT-MLP) for the rest of the paper. As the second baseline model we use plain GRUs and LSTMs that have the same size of hidden layer as their TT pendants. We follow (Liu et al., 2013) and perform for each experimental setting a 5-fold cross validation with mutual exclusive data splits. The mean and standard deviation of the prediction accuracy scores are reported in Tab. 2.

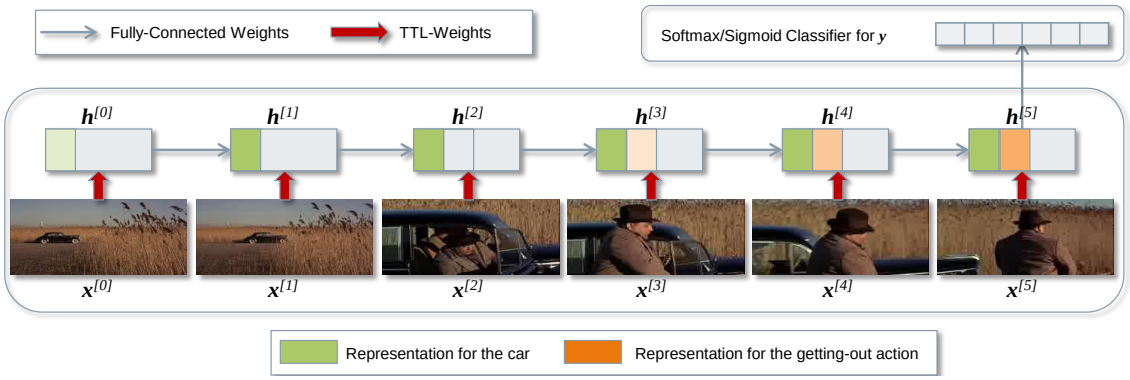


Figure 2: Architecture of the proposed model based on TT-RNN (For illustrative purposes we only show 6 frames): A softmax or sigmoid classifier built on the last hidden layer of a TT-RNN. We hypothesize that the RNN can be encouraged to aggregate the representations of different shots together and produce a global representation for the whole sequence.

Table 2: Experimental Results on UCF11 Dataset. We report i) the accuracy score, ii) the number of parameters involved in the input-to-hidden mapping in respective models and iii) the average runtime of each training epoch. The models were trained on a Quad core Intel®Xeon®E7-4850 v2 2.30GHz Processor to a maximum of 100 epochs

	Accuracy	# Parameters	Runtime
TT-MLP	0.427 ± 0.045	7,680	902s
GRU	0.488 ± 0.033	44,236,800	7,056s
LSTM	0.492 ± 0.026	58,982,400	8,892s
TT-GRU	$\mathbf{0.813 \pm 0.011}$	3,232	1,872s
TT-LSTM	0.796 ± 0.035	3,360	2,160s

The standard LSTM and GRU do not show large improvements compared with the TT-MLP model. The TT-LSTM and TT-GRU, however, do not only compress the weight matrix from over 40 millions to 3 thousands, but also significantly improve the classification accuracy. It seems that plain LSTM and GRU are not adequate to model such high-dimensional sequential data because of the large weight matrix from input to hidden layer. Compared to some latest state-of-the-art performances in Tab. 3, our model —simple as it is— shows accuracy scores second to (Sharma et al., 2015), which uses pre-trained GoogLeNet CNNs plus 3-fold stacked LSTM with attention mechanism. Please note that a GoogLeNet CNN alone consists of over 6 million parameters (Szegedy et al., 2015). In term of runtime, the plain GRU and LSTM took on average more than 8 and 10 days to train, respectively; while the TT-GRU and TT-LSTM both approximately 2 days. Therefore please note the TTL reduces the training time by a factor of 4 to 5 on these commodity hardwares.

Table 3: State-of-the-art results on the UCF11 Dataset, in comparison with our best model. Please note that there was an update of the data set on 31th December 2011. We therefore only consider works posterior to this date.

Original: (Liu et al., 2009)	0.712
(Liu et al., 2013)	0.761
(Hasan & Roy-Chowdhury, 2014)	0.690
(Sharma et al., 2015)	**0.850**
Our best model (TT-GRU)	0.813

Hollywood2 Data (Marszałek et al., 2009)

The Hollywood2 dataset contains video clips from 69 movies, from which 33 movies serve as training set and 36 movies as test set. From these movies 823 training clips and 884 test clips are generated and each clip is assigned one or multiple of 12 action labels such as answering the phone, driving a car, eating or fighting a person. This data set is much more realistic and challenging since the same action could be performed in totally different style in front of different background in different movies. Furthermore, there are often montages, camera movements and zooming within a single clip.

The original frame sizes of the videos vary, but based on the majority of the clips we generate frames of size 234×100, which corresponds to the Anamorphic Format, at fps of 12. The length of training sequences varies from 29 to 1079 with an average of 134.8; while the length of test sequences varies from 30 to 1496 frames with an average of 143.3.

The input dimension at each time step, being $234 \times 100 \times 3 = 70200$, is factorized as $10 \times 18 \times 13 \times 30$. The hidden layer is still $4 \times 4 \times 4 \times 4 = 256$ and the Tensor-Train ranks are $[1, 4, 4, 4, 1]$. Since each clip might have more

Figure 4: Two samples of frame sequences from the Hollywood2 dataset. The first sequence (row 1 and 2) belongs to the class of sitting down; the second sequence (row 3 and 4) has two labels: running and fighting person.

than one label (multi-class multi-label problem) we implement a logistic activated classifier for each class on top of the last hidden layer. Following (Marszałek et al., 2009) we measure the performances using Mean Average Precision across all classes, which corresponds to the Area-Under-Precision-Recall-Curve.

As before we conduct experiments on this dataset using the plain LSTM, GRU and their respective TT modifications. The results are presented in in Tab. 4 and state-of-the-art in Tab. 5.

Table 4: Experimental Results on Hollywood2 Dataset. We report i) the Mean Average Precision score, ii) the number of parameters involved in the input-to-hidden mapping in respective models and iii) the average runtime of each training epoch. The models were trained on an NVIDIA Tesla K40c Processor to a maximum of 500 epochs.

	MAP	# Parameters	Runtime
TT-MLP	0.103	4,352	16s
GRU	0.249	53,913,600	106s
LSTM	0.108	71,884,800	179s
TT-GRU	0.537	2,944	96s
TT-LSTM	**0.546**	3,104	102s

(Fernando et al., 2015) and (Jain et al., 2013) use improved trajectory features with Fisher encoding (Wang & Schmid, 2013) and Histogram of Optical Flow (HOF) features (Laptev et al., 2008), respectively, and achieve so far the best score. (Sharma et al., 2015) and (Fernando & Gould, 2016) provide best scores achieved with Neural Network models but only the latter applies end-to-end training. To this end, the TT-LSTM model provides the second best score in general and the best score with Neural Network models, even though it merely replaces the input-to-hidden mapping with a TTL. Please note the large difference between the plain LSTM/GRU and the TT-LSTM/GRU, which highlights the significant performance improvements the Tensor-Train Layer contributes to the RNN models.

It is also to note that, although the plain LSTM and GRU consist of up to approximately 23K as many parameters as their TT modifications do, the training *time* does not reflect such discrepancy due to the good parallelization power of GPUs. However, the obvious difference in their training *qualities* confirms that training larger models may require larger amounts of data. In such cases, powerful hardwares are no guarantee for successful training.

Table 5: State-of-the-art Results on Hollywood2 Dataset, in comparison with our best model.

Original: (Marszałek et al., 2009)	0.326
(Le et al., 2011)	0.533
(Jain et al., 2013)	0.542
(Sharma et al., 2015)	0.439
(Fernando et al., 2015)	**0.720**
(Fernando & Gould, 2016)	0.406
Our best model (TT-LSTM)	0.546

Youtube Celebrities Face Data (Kim et al., 2008)
This dataset consists of 1910 Youtube video clips of 47 prominent individuals such as movie stars and politicians. In the simplest cases, where the face of the subject is visible as a long take, a mere frame level classification would suffice. The major challenge, however, is posed by the fact that some videos involve zooming and/or changing the angle of view. In such cases a single frame may not provide enough information for the classification task and we believe it is advantageous to apply RNN models that can aggregate frame level information over time.

Figure 5: Two samples of frame sequences from the Youtube Celebrities Face dataset. The two rows belong to the classes of Al Pacino and Emma Thompson.

The original frame sizes of the videos vary but based on the majority of the clips we generate frames of size 160 $\times$ 120 at fps of 12. The retrieved sequences have lengths varying from 2 to 85 with an average of 39.9. The input dimension at each time step is $160 \times 120 \times 3 = 57600$ which is factorized as $4 \times 20 \times 20 \times 36$, the hidden layer is again $4 \times 4 \times 4 \times 4 = 256$ and the Tensor-Train ranks are

$[1, 4, 4, 4, 1]$.

Table 6: Experimental Results on Youtube Celebrities Face Dataset. We report i) the Accuracy score, ii) the number of parameters involved in the input-to-hidden mapping in respective models and iii) the average runtime of each training epoch. The models were trained on an NVIDIA Tesla K40c Processor to a maximum of 100 epochs.

	Accuracy	# Parameters	Runtime
TT-MLP	0.512 ± 0.057	3,520	14s
GRU	0.342 ± 0.023	38,880,000	212s
LSTM	0.332 ± 0.033	51,840,000	253s
TT-GRU	$\mathbf{0.800 \pm 0.018}$	3,328	72s
TT-LSTM	0.755 ± 0.033	3,392	81s

As expected, the baseline of TT-MLP model tends to perform well on the simpler video clips where the position of the face remains less changed over time, and can even outperform the plain GRU and LSTM. The TT-GRU and TT-LSTM, on the other hand, provide accuracy very close to the best state-of-the-art model (Tab. 7) using Mean Sequence Sparse Representation-based Classification (Ortiz et al., 2013) as feature extraction.

Table 7: State-of-the-art Results on Youtube Celebrities Face Dataset, in comparison with our best model.

Original: (Kim et al., 2008)	0.712
(Harandi et al., 2013)	0.739
(Ortiz et al., 2013)	**0.808**
(Faraki et al., 2016)	0.728
Our best model (TT-GRU)	0.800

Experimental Settings
We applied 0.25 Dropout (Srivastava et al., 2014) for both input-to-hidden and hidden-to-hidden mappings in plain GRU and LSTM as well as their respective TT modifications; and 0.01 ridge regularization for the single-layered classifier. The models were implemented in Theano (Bastien et al., 2012) and deployed in Keras (Chollet, 2015). We used the Adam (Kingma & Ba, 2014) step rule for the updates with an initial learning rate 0.001.

5. Conclusions and Future Work

We proposed to integrate Tensor-Train Layers into Recurrent Neural Network models including LSTM and GRU, which enables them to be trained end-to-end on high-dimensional sequential data. We tested such integration on three large-scale realistic video datasets. In comparison to the plain RNNs, which performed very poorly on these video datasets, we could empirically show that the integration of the Tensor-Train Layer alone significantly improves the modeling performances. In contrast to related works that heavily rely on deep and large CNNs, one advantage of our classification model is that it is simple and lightweight, reducing the number of free parameters from tens of millions to thousands. This would make it possible to train and deploy such models on commodity hardware and mobile devices. On the other hand, with significantly less free parameters, such tensorized models can be expected to be trained with much less labeled data, which are quite expensive in the video domain.

More importantly, we believe that our approach opens up a large number of possibilities to model high-dimensional sequential data such as videos using RNNs directly. In spite of its success in modeling other sequential data such as natural language, music data etc., RNNs have not been applied to video data in a fully end-to-end fashion, presumably due to the large input-to-hidden weight mapping. With TT-RNNs that can directly consume video clips on the pixel level, many RNN-based architectures that are successful in other applications, such as NLP, can be transferred to modeling video data: one could implement an RNN autoencoder that can learn video representations similar to (Srivastava et al., 2015), an Encoder-Decoder Network (Cho et al., 2014) that can generate captions for videos similar to (Donahue et al., 2015), or an attention-based model that can learn on which frame to allocate the attention in order to improve the classification.

We believe that the TT-RNN provides a fundamental building block that would enable the transfer of techniques from fields, where RNNs have been very successful, to fields that deal with very high-dimensional sequence data –where RNNs have failed in the past.

The source codes of our TT-RNN implementations and all the experiments in Sec. 4 are publicly available at `https://github.com/Tuyki/TT_RNN`. In addition, we also provide codes of unit tests, simulation studies as well as experiments performed on the HMDB51 dataset (Kuehne et al., 2011).

References

Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Bergstra, James, Goodfellow, Ian J., Bergeron, Arnaud, Bouchard, Nicolas, and Bengio, Yoshua. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

Bingham, Ella and Mannila, Heikki. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 245–250. ACM, 2001.

Cho, Kyunghyun, Van Merriënboer, Bart, Bahdanau, Dzmitry, and Bengio, Yoshua. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Chollet, François. Keras: Deep learning library for theano and tensorflow. https://github.com/fchollet/keras, 2015.

Donahue, Jeffrey, Anne Hendricks, Lisa, Guadarrama, Sergio, Rohrbach, Marcus, Venugopalan, Subhashini, Saenko, Kate, and Darrell, Trevor. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2625–2634, 2015.

Faraki, Masoud, Harandi, Mehrtash T, and Porikli, Fatih. Image set classification by symmetric positive semi-definite matrices. In *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on*, pp. 1–8. IEEE, 2016.

Feichtenhofer, Christoph, Pinz, Axel, and Wildes, Richard. Spatiotemporal residual networks for video action recognition. In *Advances in Neural Information Processing Systems*, pp. 3468–3476, 2016.

Fernando, Basura and Gould, Stephen. Learning end-to-end video classification with rank-pooling. In *Proc. of the International Conference on Machine Learning (ICML)*, 2016.

Fernando, Basura, Gavves, Efstratios, Oramas, Jose M, Ghodrati, Amir, and Tuytelaars, Tinne. Modeling video evolution for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5378–5387, 2015.

Garipov, Timur, Podoprikhin, Dmitry, Novikov, Alexander, and Vetrov, Dmitry. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

Han, Song, Pool, Jeff, Tran, John, and Dally, William. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.

Harandi, Mehrtash, Sanderson, Conrad, Shen, Chunhua, and Lovell, Brian C. Dictionary learning and sparse coding on grassmann manifolds: An extrinsic solution. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3120–3127, 2013.

Hasan, Mahmudul and Roy-Chowdhury, Amit K. Incremental activity modeling and recognition in streaming videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 796–803, 2014.

Jain, Mihir, Jegou, Herve, and Bouthemy, Patrick. Better exploiting motion for better action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2555–2562, 2013.

Kambhatla, Nandakishore and Leen, Todd K. Dimension reduction by local principal component analysis. *Neural computation*, 9(7):1493–1516, 1997.

Karpathy, Andrej, Toderici, George, Shetty, Sanketh, Leung, Thomas, Sukthankar, Rahul, and Fei-Fei, Li. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

Kim, Minyoung, Kumar, Sanjiv, Pavlovic, Vladimir, and Rowley, Henry. Face tracking and recognition with visual constraints in real-world videos. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8. IEEE, 2008. URL http://seqam.rutgers.edu/site/index.php?option=com_content&view=article&id=64&Itemid=80.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Koutnik, Jan, Greff, Klaus, Gomez, Faustino, and Schmidhuber, Juergen. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Kuehne, H., Jhuang, H., Garrote, E., Poggio, T., and Serre, T. HMDB: a large video database for human motion recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011.

Laptev, Ivan, Marszalek, Marcin, Schmid, Cordelia, and Rozenfeld, Benjamin. Learning realistic human actions from movies. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8. IEEE, 2008.

Le, Quoc V, Zou, Will Y, Yeung, Serena Y, and Ng, Andrew Y. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 3361–3368. IEEE, 2011.

Lebedev, Vadim, Ganin, Yaroslav, Rakhuba, Maksim, Oseledets, Ivan, and Lempitsky, Victor. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

Liu, Dianting, Shyu, Mei-Ling, and Zhao, Guiru. Spatial-temporal motion information integration for action detection and recognition in non-static background. In *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*, pp. 626–633. IEEE, 2013.

Liu, Jingen, Luo, Jiebo, and Shah, Mubarak. Recognizing realistic actions from videos "in the wild". In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 1996–2003. IEEE, 2009. URL http://crcv.ucf.edu/data/UCF_YouTube_Action.php.

Maas, Andrew L, Daly, Raymond E, Pham, Peter T, Huang, Dan, Ng, Andrew Y, and Potts, Christopher. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 142–150. Association for Computational Linguistics, 2011.

Marszałek, Marcin, Laptev, Ivan, and Schmid, Cordelia. Actions in context. In *IEEE Conference on Computer Vision & Pattern Recognition*, 2009. URL http://www.di.ens.fr/~laptev/actions/hollywood2/.

Novikov, Alexander, Podoprikhin, Dmitrii, Osokin, Anton, and Vetrov, Dmitry P. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pp. 442–450, 2015.

Ortiz, Enrique G, Wright, Alan, and Shah, Mubarak. Face recognition in movie trailers via mean sequence sparse representation-based classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3531–3538, 2013.

Oseledets, Ivan V. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

Sharma, Shikhar, Kiros, Ryan, and Salakhutdinov, Ruslan. Action recognition using visual attention. *arXiv preprint arXiv:1511.04119*, 2015.

Simonyan, Karen and Zisserman, Andrew. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, pp. 568–576, 2014.

Srivastava, Nitish, Hinton, Geoffrey E, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

Srivastava, Nitish, Mansimov, Elman, and Salakhutdinov, Ruslan. Unsupervised learning of video representations using lstms. *CoRR, abs/1502.04681*, 2, 2015.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

Wang, Heng and Schmid, Cordelia. Action recognition with improved trajectories. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3551–3558, 2013.

Xingjian, SHI, Chen, Zhourong, Wang, Hao, Yeung, Dit-Yan, Wong, Wai-Kin, and Woo, Wang-chun. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pp. 802–810, 2015.

Ye, Jieping, Janardan, Ravi, and Li, Qi. Gpca: an efficient dimension reduction scheme for image compression and retrieval. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 354–363. ACM, 2004.

Yue-Hei Ng, Joe, Hausknecht, Matthew, Vijayanarasimhan, Sudheendra, Vinyals, Oriol, Monga, Rajat, and Toderici, George. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4694–4702, 2015.

Zhang, Jun, Yan, Yong, and Lades, Martin. Face recognition: eigenface, elastic matching, and neural nets. *Proceedings of the IEEE*, 85(9):1423–1435, 1997.

Chapter 5

Tensor-Train RNNs in Modeling Sequential EHR for Survival Prediction

Modeling Progression Free Survival in Breast Cancer with Tensorized Recurrent Neural Networks and Accelerated Failure Time Models

Yinchong Yang YINCHONG.YANG@SIEMENS.COM
Ludwig Maximilians University Munich
Siemens AG, Corporate Technology, Munich

Peter A. Fasching PETER.FASCHING@UK-ERLANGEN.DE
Department of Gynecology and Obstetrics,
University Hospital Erlangen, Erlangen

Volker Tresp VOLKER.TRESP@SIEMENS.COM
Ludwig Maximilians University Munich
Siemens AG, Corporate Technology, Munich

Abstract

In this work we attempt to predict the progression-free survival time of metastatic breast cancer patients by combining state-of-the-art deep learning approaches with traditional survival analysis models. In order to tackle the challenge of sequential clinical records being both high-dimensional and sparse, we propose to apply a tensorized recurrent neural network architecture to extract a latent representation from the entire patient history. We use this as the input to an Accelerated Failure Time model that predicts the survival time. Our experiments, conducted on a large real-world clinical dataset, demonstrate that the tensorized recurrent neural network largely reduces the number of weight parameters and the training time. It also achieves modest improvements in prediction, in comparison with state-of-the-art recurrent neural network models enhanced with event embeddings.

1. Introduction

With the increasing availability of Electronic Health Records (EHR) data in clinics, there is a growing interest in predicting treatment prescription and individual patient outcome by extracting information from these data using advanced analytics approaches. Especially the latest success of deep learning in image and natural language processing has encouraged the application of these state-of-the-art techniques in modeling clinical data as well. Convolutional Neural Networks (CNNs), particularly the deeper architectures with multiple layers, turn out to be capable of not only handling natural images (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; Szegedy et al., 2015), but also medical imaging data for, e.g., segmentation and captioning (Kayalibay et al., 2017; Shin et al., 2016; Kisilev et al., 2011). On the other hand, due to the fact that natural language and medical records share the same sequential nature, Recurrent Neural Networks (RNNs), which have proven to be powerful in language modeling (Kim et al., 2015; Mikolov, 2012) and machine translation (Sutskever et al., 2014; Cho et al., 2014), are more frequently applied to medical event data to predict, e.g., medication prescription and patient endpoints, with promising per-

©2017.

formances (Choi et al., 2016, 2015). The major advantage of RNNs in handling medical records lies in their ability to handle sequential inputs of variable lengths in a more generic way than sliding window approaches (Bengio et al., 2003; Esteban et al., 2015).

In language modeling, the input to the RNN model is typically a word (or even character) embedding in form of a dense and real vector that represents the word in a latent space. The embedding idea has inspired its application on medical event data such as Choi et al. (2015); Esteban et al. (2016); Choi et al. (2016); the reason is that the sequential medical records are often of categorical type and, after binary-coding, the derived feature vector can become very sparse in a high dimensional space. Choi et al. (2015) terms such type of input features as a multi-hot vector and show that they are suboptimal to serve as direct inputs to plain RNN models. The embedding layer, though reducing the size of the input-to-hidden weight matrix in an RNN model, still requires parameters determined by the dimension of the raw multi-hot input and the embedding size.

To handle the high dimensional multi-hot sequential input, we propose in this work a simpler, more efficient and direct approach by factorizing the input-to-hidden weight matrix in an RNN model based on tensor factorization. Novikov et al. (2015) first proposed to perform Tensor-Train Factorization (Oseledets, 2011) on weight matrices in Neural Networks to tackle the challenge of input redundancy, such as in fully-connected layers following a convolutional one, or simple feed-forward layers that consume raw pixel data. This tensorization is proven to be highly efficient and can significantly reduce the over-parameterization in the weights without sacrificing much of the model expressiveness. Following their work, we argue that the same challenge of redundant information in high dimensional and sequential multi-hot vectors can be addressed in a similar way by integrating the Tensor-Train factorization into RNNs, with the nice property that the factorization is learned efficiently, together with the rest of the RNN, in an end-to-end fashion.

We conduct experiments on a large real-world dataset consisting of thousands of metastatic breast cancer patients from Germany. We attempt to predict for each patient her/his Progression-Free Survival time based on her/his medical history of variable length and background information, using an Accelerated Failure Time (AFT) model in combination with the RNN models. The former (Wei, 1992) has always been a promising tool for survival analysis, especially if one wants to directly model the individual survival time instead of the hazard. Gore et al. (1984) and Bradburn et al. (2003), for instance, have demonstrated the application of AFT in modeling progression-free survival in case of breast and lung cancer, respectively.

The rest of the paper is organized as follows: in Sec. 2 we give an overview of related works that have inspired our own; in Sec. 3 we provide an introduction to our cohort and data situation; in Sec. 4 we present the novel techniques of our model in detail and provide experimental results in Sec. 5. Finally, Sec. 6 provides a conclusion and an outlook on future works.

2. Related Work

Handling sequential EHR data. Due to the sequential nature of EHR data, there have been recently multiple promising works studying clinical events as sequential data. Many of them were inspired by works in natural language modeling, since sentences can

be easily modeled as sequences of signals. Esteban et al. (2015) adjusted a language model based on the sliding window technique in Bengio et al. (2003), taking into account a fixed number of events in the past. This model was extended in Esteban et al. (2016) by replacing the sliding window with RNNs, which improved the predictions for prescriptions decision and endpoints. Lipton et al. (2015) applied LSTM to perform diagnosis prediction based on sequential input. A related approach with RNNs can also be found in Choi et al. (2015) to predict diagnosis and medication prescriptions. This RNN implementation was further augmented with neural attention mechanism in Choi et al. (2016), which did not only show promising performances but also improved the interpretability of the model.

RNN for sequence classification and regression. The RNN models in these works were implemented in a many-to-many fashion. That is to say, at each time step the RNN generates a prediction as output, since the target in these works is provided at every time step. In our work, on the other hand, cancer progression is not expected to be observed regularly. Consequently, we implement many-to-one RNN models that consume a sequence of input vectors, and generate only one output vector. This setting can be found in a variety of sequence classification/regression tasks. Koutnik et al. (2014) used such RNN architectures to classify spoken words and handwriting as sequences. RNNs have also been applied to classify the sentiment of sentences such as in the IMDB reviews dataset (Maas et al., 2011). The application of RNNs in the many-to-one fashion can also be seen as to encode a sequence of variable length into one fixed-size representation (Sutskever et al., 2014) and then to perform prediction as decoding based on this representation.

Tensor-Train Factorization and Tensor-Train Layer. The *Tensor-Train Factorization* (TTF) was first introduced by Oseledets (2011) as a tensor factorization model with the advantage of being capable of scaling the factorization to an arbitrary number of dimensions. Novikov et al. (2015) showed that one could reshape a fully connected layer into a high dimensional tensor, which is to be factorized using TTF, and referred to it as a *Tensor-Train Layer* (TTL). This idea was applied to compress very large weight matrices in deep Neural Networks where the entire model was trained end-to-end. In these experiments they compressed fully connected layers on top of convolutional layers, and also proved that a Tensor-Train layer can directly consume pixels of image data such as CIFAR-10, achieving the best result among all known non-convolutional layers. Then in Garipov et al. (2016) it was shown that even the convolutional layers themselves can be compressed with TTLs. Yang et al. (2017) first applied TTF to the input-to-hidden layer in RNN variants. They showed that such modification not only reduced the number of weight parameters in orders of magnitude, but also could boost prediction accuracy in classification of real-world video clips, which typically involve extremely large input dimensions.

3. Cohort

3.1 Data Extraction

In Germany, approximately 70,000 women suffer from breast cancer and the mortality is approximately 33% every year (Kaatsch et al., 2013; Rauh and Matthias, 2008). In many of these cases, it is the progression, especially the metastasis of the cancer cells to vital organs, that actually causes the patient's death. Our dataset, provided by the PRAEGNANT study network (Fasching et al., 2015), was collected on patients suffering from metastatic

breast cancer and warehoused in the secuTrial®, which is a relational database system. After querying and pre-processing, we could retrieve information on 4,357 valid *patient cases*, which we define as a patient-time pair, i.e., at that specific time a metastasis and/or recurrence is observed on that patient. The first patient was recruited in 2014 and the currently last patient in 2016, but their earliest medical records date back to 1961.

3.2 Feature Processing

There are two classes of patient information that are potentially relevant for modeling the PFS time:

First, the *static* information includes 1) basic patient information, 2) information on the primary tumor and 3) history of metastasis before entering the study. In total we observe 26 features of binary, categorical or real types. We perform binary-coding on the former both cases and could extract for each patient case i a static feature vector denoted with $\boldsymbol{m}_i \in \mathbb{R}^{118}$.

The *sequential* information includes 4) local recurrences, 5) metastasis 6) clinical visits 7) radiotherapies, 8) systemic therapies and 9) surgeries. These are time-stamped clinical events observed on each patient throughout time. In total we have 26 sequential features of binary or categorical nature. Binary-coded, they yield for a patient case i at time step t a feature vector $\boldsymbol{x}_i^{[t]} \in \{0,1\}^{196}$. We denote the whole sequence of events for this patient case i up to her/his last observation at T_i before progression (in form of either local recurrences or metastasis) using a set of $\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i}$. The length of the sequences vary from 1 to 35 and is on average 7.54.

Due to the binary-coding, each $\boldsymbol{m}_i$ and $\boldsymbol{x}_i^{[t]}$ yield on average sparsities of 0.88 and 0.97, respectively, for each i and t.

3.3 Distribution of the Target Variable

We denote the number of days till the next recorded progression using z_i, and assume it to be a realization of a random variable Z_i to serve as the target variable of our model.

In total, we attempt to model the distribution of PFS time as Generalized Regression Model (GRM):

$$Z_i \overset{i.i.d.}{\sim} \mathcal{F}(\Theta_i) \quad \text{where} \quad \Theta_i = g^{-1}(\boldsymbol{\eta}_i^{[T_i]}). \tag{1}$$

where Z_i is a variable independently conditioned on some features that represent the patient case, where g^{-1} is analogous to the inverse link function in GRMs and maps the time dependent input $\boldsymbol{\eta}_i^{[T_i]}$ to the set of distribution parameters Θ_i.

We would assume Z_i to be Log-Normal distributed and specify Eq. (1) to be:

$$Z_i \overset{i.i.d.}{\sim} \log\mathcal{N}(\mu_i, \sigma_i) \text{ with } \mu_i = g^{-1}(\boldsymbol{\eta}_i^{[T_i]}) = g^{-1}\left(f\left(\boldsymbol{m}_i, \{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i}\right)\right). \tag{2}$$

In the section that follows we elaborate in detail the construction of the function f, which maps the static and sequential features of a patient case into a latent representation $\boldsymbol{\eta}_i^{[T_i]}$ to serve as the input to the AFT model, a special case of GRM.

4. Methods

In this section, we first give an introduction to the Tensor-Train Layer; we then use this to replace the weight matrix mapping from the input vector to hidden state in RNN models. Thereafter, we briefly review the Accelerated Failure Time model that consumes the latent patient representation produced by the Tensor-Train RNN.

4.1 Tensor-Train Recurrent Neural Networks

Tensor-Train Factorization of a Feed-Forward Layer Novikov et al. (2015) showed that the weight matrix $\boldsymbol{W}$ of a fully-connected feed-forward layer $\hat{\boldsymbol{y}} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ can be factorized using Tensor-Train (Oseledets, 2011), where the factorization and the weights are learned simultaneously. This modification is especially effective, if the layer is exposed to high-dimensional input with redundant information, such as pixel input and feature maps produced by convolutional layers. In this work, we show that the challenge of multi-hot representation of sequential EHR data can also be handled using TTL by integrating it into RNNs.

A fully-connected feed-forward layer, in form of:

$$\hat{\boldsymbol{y}}(j) = \sum_{i=1}^{M} \boldsymbol{W}(i,j) \cdot \boldsymbol{x}(i) + \boldsymbol{b}(j), \ \forall j \in [1, N] \text{ with } \ \boldsymbol{x} \in \mathbb{R}^M, \ \boldsymbol{y} \in \mathbb{R}^N, \tag{3}$$

can be equivalently rewritten as

$$\begin{aligned} \widehat{\boldsymbol{\mathcal{Y}}}(j_1, j_2, ..., j_d) = \sum_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} ... \sum_{i_d=1}^{m_d} \boldsymbol{\mathcal{W}}((i_1, j_1), (i_2, j_2), ..., (i_d, j_d)) \cdot \boldsymbol{\mathcal{X}}(i_1, i_2, ..., i_d) \\ + \boldsymbol{\mathcal{B}}(j_1, j_2, ..., j_d), \end{aligned} \tag{4}$$

so long as the input and output dimensions can be factorized as $M = \prod_{k=1}^{d} m_k, \quad N = \prod_{k=1}^{d} n_k$. Therefore, the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are reshaped into two tensors with the same number of dimensions: $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m_1 \times m_2 \times ... \times m_d}, \boldsymbol{\mathcal{Y}} \in \mathbb{R}^{n_1 \times n_2 \times ... \times n_d}$, respectively, and the mapping function becomes $\mathbb{R}^{m_1 \times m_2 \times ... \times m_d} \rightarrow \mathbb{R}^{n_1 \times n_2 \times ... \times n_d}$. Though mathematically equivalent, Eq. 4 represents a more general description of Eq. 3 and, more importantly, provides a high dimensional weight *tensor* of $\boldsymbol{\mathcal{W}}$ that can be factorized using TTF:

$$\widehat{\boldsymbol{\mathcal{W}}}((i_1, j_1), (i_2, j_2), ..., (i_d, j_d)) \overset{TTF}{=} \boldsymbol{\mathcal{G}}_1^*(i_1, j_1, :, :) \ \boldsymbol{\mathcal{G}}_2^*(i_2, j_2, :, :) \ ... \ \boldsymbol{\mathcal{G}}_d^*(i_d, j_d, :, :), \tag{5}$$

where a $\boldsymbol{\mathcal{G}}_k^* \in \mathbb{R}^{m_k \times n_k \times r_{k-1} \times r_k}$ is termed as a core tensor, specified by ranks r_k for $k \in [0, d]$. Now instead of explicitly storing the full tensor $\boldsymbol{\mathcal{W}}$ of size $\prod_{k=1}^{d} m_k \cdot n_k = M \cdot N$, we only store its TT-format, i.e., the set of low-rank core tensors $\{\boldsymbol{\mathcal{G}}_k^*\}_{k=1}^{d}$ which can approximately reconstruct $\boldsymbol{\mathcal{W}}$.

For the rest of the paper, we denote a fully-connected layer of $\hat{\boldsymbol{y}} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, whose weight matrix $\boldsymbol{W}$ is factorized with TTF as $\hat{\boldsymbol{y}} = TTL(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{x})$, or $TTL(\boldsymbol{W}, \boldsymbol{x})$, if no bias is required.

Tensor-Train RNN In this work we investigate the challenge of modeling high dimensional sequential data with RNNs. For this reason, we factorize the matrix mapping from

the input to the hidden state with a TTL as in Yang et al. (2017). More specifically, in case of LSTM, a particular form of RNN by Hochreiter and Schmidhuber (1997); Gers et al. (2000) and GRU, another variant by Chung et al. (2014), we TT-factorize the matrices that map from the input vector to the gating units as in Eq. (6):

$$
\begin{aligned}
&\text{TT-GRU:} && \text{TT-LSTM:} \\
&\boldsymbol{r}^{[t]} = \text{sig}(TTL(\boldsymbol{W}^r, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^r \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^r) && \boldsymbol{k}^{[t]} = \text{sig}(TTL(\boldsymbol{W}^k, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^k \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^k) \\
&\boldsymbol{z}^{[t]} = \text{sig}(TTL(\boldsymbol{W}^z, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^z \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^z) && \boldsymbol{f}^{[t]} = \text{sig}(TTL(\boldsymbol{W}^f, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^f \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^f) \\
&\boldsymbol{d}^{[t]} = \tanh(TTL(\boldsymbol{W}^d, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^d (\boldsymbol{r}^{[t]} \circ \boldsymbol{h}^{[t-1]})) && \boldsymbol{o}^{[t]} = \text{sig}(TTL(\boldsymbol{W}^o, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^o \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^o) \\
&\boldsymbol{h}^{[t]} = (1 - \boldsymbol{z}^{[t]}) \circ \boldsymbol{h}^{[t-1]} + \boldsymbol{z}^{[t]} \circ \boldsymbol{d}^{[t]}, && \boldsymbol{g}^{[t]} = \tanh(TTL(\boldsymbol{W}^g, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^g \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^g) \\
& && \boldsymbol{c}^{[t]} = \boldsymbol{f}^{[t]} \circ \boldsymbol{c}^{[t-1]} + \boldsymbol{k}^{[t]} \circ \boldsymbol{g}^{[t]} \\
& && \boldsymbol{h}^{[t]} = \boldsymbol{o}^{[t]} \circ \tanh(\boldsymbol{c}^{[t]}).
\end{aligned} \tag{6}
$$

For the sake of simplicity, we denote a many-to-one RNN (either with or without Tensor-Train, either GRU or LSTM) using a function ω: $\boldsymbol{h}_i^{[T_i]} = \omega(\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i})$, where $\boldsymbol{h}_i^{[T_i]}$ is the last hidden state as in Eq. (6).

In order to also take into account the static features (Esteban et al., 2016) such as patient background and primary tumor, we concatenate the last hidden state of the RNN with the latent representation of the static features as $\boldsymbol{\eta}_i^{[T_i]} = [\boldsymbol{h}_i^{[T_i]}, \boldsymbol{q}_i]$ with $\boldsymbol{q}_i = \psi(\boldsymbol{V}\boldsymbol{m}_i)$, where $\boldsymbol{V}$ is a standard trainable weight matrix and ψ denotes a non-linear activation function. Finally, we have specified the function f with respect to Eq. (2) as:

$$
g(\mu_i) = \boldsymbol{\eta}_i^{[T_i]} = f\left(\boldsymbol{m}_i, \{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i}\right) = \left[\psi(\boldsymbol{V}\boldsymbol{m}_i),\ \omega(\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i})\right]. \tag{7}
$$

The vector $\boldsymbol{\eta}_i^{[T_i]}$ represents the static patient information as well as the medical history of patient i up to time step T_i. In the context of GRM-like models, $\boldsymbol{\eta}_i^{[T_i]}$ would be the raw *covariates*, while in our case, it is a more abstract *latent representation* generated from a variety of raw and potentially less structured features. This point of view provides us with an interface between representation learning and the GRM models.

The vector $\boldsymbol{\eta}_i^{[T_i]}$ also functions as an abstract patient profile that represents all relevant clinical information in a latent vector space, where patients with similar background information and medical history would be placed in a specific neighborhood. This very characteristics of the latent vector space is key to the latest success of deep or representation learning, because it facilitates the classification and regression models built on top of it, which is, in our case, an AFT model that is presented as follows.

4.2 Accelerated Failure Time Model

The AFT model is a GRM-like parametric regression that attempts to capture the influence that the features in $\boldsymbol{\eta}_i$ have on a variable Z_i, which describes the survival time till an event is observed:

$$
Y_i = \boldsymbol{\eta}_i^T \boldsymbol{\beta} + R_i, \text{ with } Y_i = \log(Z_i), \tag{8}
$$

where $\boldsymbol{\beta}$ is the weight vector and $R_i \overset{i.i.d.}{\sim} \mathcal{D}_R$ is the residual whose distribution can be specified by $\mathcal{D}_R$. Common choices for $\mathcal{D}_R$ are Normal, Extreme Value and Logistic distributions. The variable Z would correspondingly be Log-Normal, Weibull/Exponential and

Log-Logistic distributed, respectively. As Eq. (8) suggests, an AFT model assumes that the covariates have a multiplicative effect on the survival time, 'accelerating' —either positively or negatively— the baseline time till which the event of interest occurs. To see that one only has to rewrite Eq. (8) as $Z_i = Z_0 \exp\left(\boldsymbol{\eta}_i^T \boldsymbol{\beta}\right)$ with $Z_0 = \exp\left(R_i\right)$, so that the baseline survival time Z_0 is accelerated to a factor of $\exp\{\boldsymbol{\eta}_i^T \boldsymbol{\beta}\}$. In other words, if one covariate j increases by a factor of δ, the failure time is to be accelerated by a factor of $\exp\{\delta\beta_j\}$, so long as all other covariates remain the same. The Proportional Hazard Cox Regression, on the other hand, assumes such a multiplicative effect over the hazard.

We are specifically assuming that the target variable Z_i follows a Log-Normal distribution and that, implicitly, the residuals are normal distributed, i.e., $R_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma_i)$. One could therefore plug Eq. (8) into the normal distribution assumption and have $\frac{\log(Z_i) - \boldsymbol{\eta}_i^T \boldsymbol{\beta}}{\sigma_i} \overset{i.i.d.}{\sim} \mathcal{N}(0, 1)$, which allows us to perform training with the Mean Squared Logarithmic Error.

5. Experiments

5.1 Experimental Details and Evaluation Approach

We conduct 5-fold cross-validations by splitting the dataset into proportions of 0.8 / 0.2 disjoint subsets for training / test tasks. We train our models with 0.25 dropout (Srivastava et al., 2014) rate for the weights in (TT-)RNNs, and 0.025 ridge-penalization in feed-forward layers, with the Adam (Kingma and Ba, 2014) step rule for 200 epochs. Since the dimension of 196 can be represented with prime factors $2^2 \times 7^2$, we experiment two different settings of $[7, 28]$ and $[4, 7, 7]$ to factorized the input dimension. The corresponding factorizations of the RNN output's dimension are $[8, 8]$ and $[4, 4, 4]$, respectively. In the first case, the Tensor-Train-Factorization becomes equivalent to a two mode PARAFAC/CP (Kolda and Bader, 2009) model, and we therefore denote GRU and LSTM with this setting as CP-GRU and CP-LSTM, respectively, for the sake of simplicity. We also experiment two different TT-ranks of 4 and 6. All models are trained with objective function of Mean Squared Logarithmic Error (Chollet, 2015). We set the size of the non-linear mapping of the static feature $\boldsymbol{q}_i$ to be 128.

Since the target Z follows a Log-Normal instead of Normal distribution, it is not appropriate to measure the results in term of Mean Squared Error (MSE) as is with usual regression models. We therefore report the more robust metric of Median Absolute Error (MAE) defined as $MAE = median_i(|z_i - \hat{z}_i|)$. Even on the logarithmic scale of Y, MSE is not a reliable metric in this case either, since the Squared Error of a $y_i = \log z_i$ would increase in Z exponentially as y_i increases. In other words, two similar Squared Errors on the logarithmic scale Y might imply totally different errors in Z. We therefore report the coefficient of determination as $R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$ on the logarithmic scale of Y.

All our models are implemented in Theano (Bastien et al., 2012) and deployed in Keras (Chollet, 2015). The experiments were conducted on a NVIDIA Tesla K40c Processor.

5.2 Prediction of Progression-Free Survival

As weak baselines we first report the performance of standard Cox and AFT Regression using the R package `survival`(Therneau, 2015; Terry M. Therneau and Patricia M. Gramb-

Model	TT-Rank	MAE	R^2	Time(sec.)	#Parameters
GRU	-	156.7 ± 6.6	0.295 ± 0.018	3,598	74,880
LSTM	-	159.5 ± 13.7	0.274 ± 0.014	5,956	99,840
Emb.+GRU	-	136.2 ± 14.8	0.635 ± 0.009	4,957	24,832
Emb.+LSTM	-	136.8 ± 11.9	0.633 ± 0.014	6,170	28,928
CP-GRU	4	135.6 ± 10.0	0.630 ± 0.018	**1,689**	1,568
	6	136.8 ± 9.3	0.623 ± 0.015	1,773	2,352
TT-GRU	4	136.5 ± 9.4	0.632 ± 0.020	1,940	**752**
	6	136.2 ± 11.4	0.634 ± 0.019	2,178	1,464
CP-LSTM	4	135.2 ± 8.4	0.637 ± 0.018	3,390	1,792
	6	133.7 ± 10.7	0.625 ± 0.023	3,530	2,688
TT-LSTM	4	140.0 ± 10.4	**0.645**± 0.025	3,729	816
	6	**132.9** ± 9.9	0.630 ± 0.017	4,050	1,560

Table 1: Experimental results: average MAE, R^2, average training time, the number of all parameters responsible for mapping the raw input to the hidden state in RNNs.

sch, 2000). The Cox Regression (in term of median survival estimate in the package) yields an average MAE score over the cross-validation of 214.5 and the AFT 208.7. The input to both models are the raw features aggregated w.r.t time axis. Such aggregation is also applied in Esteban et al. (2015) and is proven to be a reasonable alternative solution to handle time stamped features, since each entry represents the number of feature values observed up to a specific time step, though ignoring the order in which the events were observed.

We apply two further classes of baseline models: First, we expose GRU and LSTM directly to the raw sequential features and then, secondly, we add a *tanh* activated embedding layer of size 64, between the raw input feature and the RNNs, following the state-of-the-art of Choi et al. (2015). The corresponding results are presented in the first four rows in Tab. 1. Compared to aggregated sequential features, RNNs are indeed able to generate representations that facilitate the AFT model on top of them, reducing the MAE from over 200 to ca. 150. The embedding layers as input to RNNs turn out to yield even better results of 136, with less parameters but longer training time. This confirms the point made in Choi et al. (2015) that such input features of both high dimensionality and sparsity are suboptimal input to RNNs.

In further experiments we test our TT-GRU, TT-LSTM, CP-GRU and CP-LSTM implementations with different TT-ranks. Though exposed to the raw features, these tensorized RNNs yield prediction quality comparable with –and sometimes even better than– the state-of-the-art embedding technique and require on average ca. 40% of the training time and $2\% - 10\%$ of parameters. Compared with plain RNNs, they merely require 1% to 3% of the parameters.

In contrast to Novikov et al. (2015) where tensorization slightly decreased the prediction quality, we actually observe on average a modest improvement with TT-RNNs. For instance, the MAE is improved from 136.2 to 135.6 in case of GRU and from 136.8 to 132.9 in case of LSTM. Such MAE measures around 135 implies that these model can provide a prediction of PFS time with an accuracy of plus-minus four months time for the non-extreme patient cases.

$\mathcal{F}_0$	Weibull	Exponential	Log-Logistic	Log-Normal
p-value	3.7e-4	$\leq$2.2e-16	$\leq$2.2e-16	0.064

Table 2: Two-sided Kolmogorov-Smirnov Tests of the distribution of the residual under the H_0 of variable distributions $\mathcal{F}_0$.

Please note that in an RNN model, the input-to-hidden weight matrix becomes over-parameterized if the input feature turns out to be highly sparse and/or to consist of high proportion of redundant information. In earlier works, this challenge is tackled using an explicit embedding layer that transforms the raw feature into a more compact input vector to the RNNs. The TT-RNNs, on the other hand, provide an alternative solution that directly tackles the over-parameterization in the weight matrix, in that the full-sized weight matrix is constructed using a much smaller number of *'meta'* weights, i.e., the core tensors in the Tensor-Train model.

Secondly, comparing different TT-settings, it is trivial that a higher TT-rank requires more parameters and longer training time. Compared with the CP-factorization of $196 = 7 \times 28$, the real TT-factorization of $196 = 4 \times 7 \times 7$ of the weight matrix leads to a smaller number of parameters, but the difference in training time is less extreme. This can be explained with Eq. (5), which demonstrates that the number of core tensors also influences the computation complexity. More specifically, the multiplication among core tensors is strictly successive in $k = 1, 2, ..., d$ and cannot be parallelized in CPUs or GPUs. In other words, a chain of multiplication of small core tensors might, in extreme cases, take even longer to compute than the multiplication of two large matrices.

In order to verify the Log-Normal distribution assumption, we conduct Kolmogorov-Smirnov-Tests as in R Core Team (2016) on the modeling residuals. We report in Tab. 2 the p-values corresponding to alternative distribution assumptions. The hypothesis of Weibull-, Exponential and Log-Logistic-distribution can be therefore rejected with rather high significance. Since we cannot reject the Log-Normal distribution assumption even with the largest common significant level of 5%, our assumption of Z_i to be Log-Normal distributed in Eq. (1) can be verified.

5.3 Estimation of Individual Survival and Hazard Functions

Beside the PFS time, the AFT model also allows us to calculate individual survival and hazard functions for each patient case. The individual survival and hazard function can be derived from Eq. (8) and takes the form of:

$$S(Z_i = z|\boldsymbol{\eta}_i) = 1 - \Phi\left(\frac{\log(z) - \boldsymbol{\eta}_i^T\boldsymbol{\beta}}{\sigma_i}\right), \tag{9}$$

$$\lambda(Z_i = z|\boldsymbol{\eta}_i) = -\frac{\partial}{\partial z}S_i(z|\boldsymbol{\eta}_i) = \frac{\phi(\frac{\log(z)-\boldsymbol{\eta}_i^T\boldsymbol{\beta}}{\sigma_i})}{\left(1 - \Phi\left(\frac{\log(z)-\boldsymbol{\eta}_i^T\boldsymbol{\beta}}{\sigma_i}\right)\right) z\sigma_i}, \tag{10}$$

where Φ and ϕ are the cumulative distribution function and probability density function of a standard normal distribution, respectively. Here σ_i can be estimated using

$$\sigma_i \approx \hat{\sigma}_i = \mathbb{V}_{i^* \sim data_{train}}\left(\log(Z_{i^*}) - \boldsymbol{\eta}_{i^*}^T\boldsymbol{\beta}\right)^{\frac{1}{2}}, \tag{11}$$

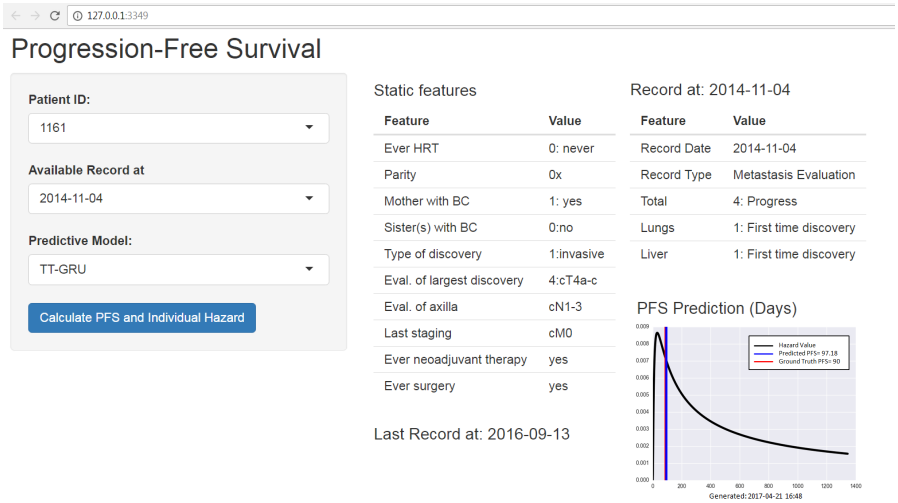


Figure 1: A prototype App for patient data querying and PFS prediction, implemented with R Shiny Chang et al. (2015), illustrated with information of a patient from our test set.

under the conditional i.i.d. assumption in Eq. (1). In a realistic application scenario in clinics, providing the individual hazard function is as important as providing physicians with a prediction of the PFS time. For the clinic we have developed a prototype interface (Fig. 1) to predict the PFS time and individual hazard function as well as to query patient information. For illustrative purposes, we calculate hazard function for a patient from test set so that, for comparison, we also mark the ground truth PFS.

6. Conclusion and Future Works

We have first applied tensorized RNNs to handle high dimensional sequential inputs in form of medical history data; Second, we showed that one can join deep/Representation Learning with GRM-like models in a Encoder-Decoder fashion (Sutskever et al., 2014), where the RNN encoder produces better representative input for the GRM-like decoder. Our empirical results demonstrate that the tensorized RNNs greatly reduce the number of parameters and the training time of the model, while retaining –if not improving– the prediction quality compared to the state-of-the-art embedding technique. We also showed that when applying an AFT model on top of RNNs, one could calculate the PFS time as well as provide physicians with individual survival and hazard functions, improving the usability of our model in realistic scenarios. In the future we would like to integrate attention mechanisms as Choi et al. (2016) into the TT-RNNs, in order to improve the model's interpretability. This would enables the physician to trace back to event(s) that have contributed most to the prediction. In this work we implicitly reshape our multi-hot input solely for computational convenience. We find it therefore necessary to explore possibilities to perform the reshaping in accordance with the actual input structure. Furthermore it seems also appealing to include other distribution assumptions for the AFT model.

References

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *journal of machine learning research*, 3(Feb):1137–1155, 2003.

Mike J Bradburn, Taane G Clark, SB Love, and DG Altman. Survival analysis part ii: multivariate data analysis-an introduction to concepts and methods. *The British Journal of Cancer*, 89(3):431, 2003.

Winston Chang, Joe Cheng, J Allaire, Yihui Xie, and Jonathan McPherson. Shiny: web application framework for r. *R package version 0.11*, 1, 2015.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Edward Choi, Mohammad Taha Bahadori, and Jimeng Sun. Doctor ai: Predicting clinical events via recurrent neural networks. *arXiv preprint arXiv:1511.05942*, 2015.

Edward Choi, Mohammad Taha Bahadori, Jimeng Sun, Joshua Kulas, Andy Schuetz, and Walter Stewart. Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. In *Advances in Neural Information Processing Systems*, pages 3504–3512, 2016.

François Chollet. Keras: Deep learning library for theano and tensorflow. `https://github.com/fchollet/keras`, 2015.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Cristóbal Esteban, Danilo Schmidt, Denis Krompaß, and Volker Tresp. Predicting sequences of clinical events by using a personalized temporal latent embedding model. In *Healthcare Informatics (ICHI), 2015 International Conference on*, pages 130–139. IEEE, 2015.

Cristóbal Esteban, Oliver Staeck, Yinchong Yang, and Volker Tresp. Predicting clinical events by combining static and dynamic information using recurrent neural networks. *arXiv preprint arXiv:1602.02685*, 2016.

P.A. Fasching, S.Y. Brucker, T.N. Fehm, F. Overkamp, W. Janni, M. Wallwiener, P. Hadji, E. Belleville, L. Häberle, F.A. Taran, D. Luftner, M.P. Lux, J. Ettl, V. Muller, H. Tesch, D. Wallwiener, and A. Schneeweiss. Biomarkers in patients with metastatic breast cancer and the praegnant study network. *Geburtshilfe Frauenheilkunde*, 75(01):41–50, 2015.

Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

Sheila M Gore, Stuart J Pocock, and Gillian R Kerr. Regression models and non-proportional hazards in the analysis of breast cancer survival. *Applied Statistics*, pages 176–195, 1984.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Peter Kaatsch, Claudia Spix, Stefan Hentschel, Alexander Katalinic, Sabine Luttmann, Christa Stegmaier, Sandra Caspritz, Josef Cernaj, Anke Ernst, Juliane Folkerts, et al. Krebs in deutschland 2009/2010. 2013.

Baris Kayalibay, Grady Jensen, and Patrick van der Smagt. Cnn-based segmentation of medical imaging data. *arXiv preprint arXiv:1701.03056*, 2017.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. *arXiv preprint arXiv:1508.06615*, 2015.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Pavel Kisilev, Eli Sason, Ella Barkan, and Sharbell Hashoul. Medical image captioning: learning to describe medical image findings using multi-task-loss cnn. 2011.

Tamara G. Kolda and Brett W. Bader. Tensor Decompositions and Applications. *SIAM Review*, 2009.

Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzell. Learning to diagnose with lstm recurrent neural networks. *arXiv preprint arXiv:1511.03677*, 2015.

Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 142–150. Association for Computational Linguistics, 2011.

Tomas Mikolov. Statistical language models based on neural networks. 2012.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pages 442–450, 2015.

Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33 (5):2295–2317, 2011.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016. URL `https://www.R-project.org/`.

Claudia Rauh and W Matthias. Interdisziplinäre s3-leitlinie für die diagnostik, therapie und nachsorge des mammakarzinoms. 2008.

Hoo-Chang Shin, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M Summers. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5):1285–1298, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

Terry M. Therneau and Patricia M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer, New York, 2000. ISBN 0-387-98784-3.

Terry M Therneau. *A Package for Survival Analysis in S*, 2015. URL `https://CRAN.R-project.org/package=survival`. version 2.38.

Lee-Jen Wei. The accelerated failure time model: a useful alternative to the cox regression model in survival analysis. *Statistics in medicine*, 11(14-15):1871–1879, 1992.

Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In *Proc. of the International Conference on Machine Learning (ICML)*, 2017.

Chapter 6

Summary of Contributions

Representation learning in knowledge graphs:

- Tensor decomposition as a neural network

In relational learning in knowledge graphs, tensor decomposition is proven to be the state-of-the-art approach. It learns latent vectors that represent entities, as well as a function to join the representation vectors to reconstruct the tensor. This approach can predict the probability of any unknown fact that involves known entities. It cannot, however, produce representation vectors for new entities without retraining the decomposition model completely.

We developed in [114] a novel mapping function that derives the latent representation of the new entities based on all available facts that involve this new entity, even when only a small fraction of such information is available. Such a mapping can be learned end-to-end in conjunction with the decomposition in a very efficient way using back propagation. We used a feed-forward neural network as the reconstruction function and a binary vector indicating all available facts about the new entity as inputs. In comparison with latent representations derived from complete retraining that would take up to multiple hours, our method showed 0.015 of decline in AUROC, but required only a few seconds of run time. This approach is also a generic one in the sense that one could as well include additional attributes on the entities into the input vector to solve the cold-start problem in recommendation systems.

Representation learning for high dimensional sequence modeling:

- Tensorized and decomposed neural networks

Sequential data also pose a considerable challenge to predictive machine learning, especially

when the sequences are of variable lengths. Specifically, in our work, we have patient data consisting of clinical events, such as cancer progression and therapy prescriptions. Each patient has a different time line and a variable number of clinical events. In other words, these patients do not share common input feature space.

Inspired by the success of recurrent neural networks in in natural language processing, such as sentiment analysis and machine translation, we apply them to generate fixed-size latent representation from these sequences of patient features. These latent vectors represent all patients in a common feature space and form the input to the prediction model of either classification or regression.

In [116], we proposed to apply the multinomial hierarchical regression model as the prediction model. Such a regression model mimics the hierarchical decision procedure and predicts the therapy prescription for each patient. Subsequently, these predictions are presented to the physicians as therapy recommendations. In comparison with our strong baseline model that aggregated the sequential features, the recurrent neural network encoders boosted the prediction quality by 16.5% in AUROC and 110.4% in AUPRC. We also showed that these latent representations did not only facilitate a regression model to consume sequential features, but also provide a means to compare patients in the latent space. Please note that this would have been infeasible w.r.t. the raw features since there is no common input feature space. Our machine learning model is thus capable of generating a list of similar patients to support each therapy prediction, and the identified similar patients proved to have received similar therapies.

We further studied the fact that these patient features are not only sequential, but also of high dimensionality and high sparsity, due to the fact that the original categorical features, such as medications and therapies, have to be binary coded.

In [117], we introduced a novel and generic solution, performing Tensor-Train decomposition [85] on the input-to-hidden layer in recurrent neural networks (GRU and LSTM). We applied these models on video data, because they can be handled as sequences of image frames. Plain GRU and LSTM are difficult to train on such data because of the extremely large weight matrix mapping from input to hidden state. The introduced Tensor-Train GRU and tensor-train LSTM, on the other hand, achieved 66.6% more accuracy, while using weights of size 5 orders of magnitude smaller. Even in comparison with state-of-the-art deep neural network solutions on the same datasets, our model was still 4 orders of magnitude smaller while being only 0.04 lower in accuracy.

In [115], we then applied these tensor-train recurrent neural networks on the patient

features. By exposing the fixed-size latent representation to the accelerated failure time model, it predicts the progression-free survival time for each patient and derives the individual survival function, based on the patient's entire medical history. The Tensor-Train RNNs used only a fraction of 0.01 of the weight parameters necessary in the plain models, reduced the training time to 29.7%, and improved the prediction quality by 15.4%. We also elaborated another advantage of learning representation: in cases where the raw features are unstructured and thus challenging for classical generalized regression models, deep learning models could be exploited to generate more informative features that are of much smaller dimensionality.

In Tab. 6.1, we summarize the models studied in this work. Following the framework defined in Def. 1 we decompose each model into a representation part and a prediction part. In comparison with the standard relational model based on tensor decomposition (first row), our proposed representation mapping model (second row) defines a new kind of input to the representation model g, namely the slices from the target tensor $\boldsymbol{\mathcal{Y}}$ itself, instead of one-hot vectors. We also modify standard RNN models by decomposing the input-to-hidden weight matrix $\boldsymbol{W}$ into a set of core tensors $\{\boldsymbol{\mathcal{G}}_k\}_{k=1}^{d}$. The hidden state of RNN models are then fed to prediction models depending on the various predictive task. Logistic regression and multinomial hierarchical regression, for instance, serves to predict therapy decisions. We also use logistic regression to predict the class of video clips. Accelerated Failure Time models is also exposed to the latent representation generated by (Tensor-Train) RNNs to perform survival modeling.

	Raw Input	Representation Model		Representation	Prediction Model	Section
Framework Eq. 1.1	$\boldsymbol{x}$	$\xi(\cdot)$	Φ	$\boldsymbol{z}$	$\eta_\Theta(\cdot)$	1
Relational Learning	$\{\boldsymbol{e}_i^I, \boldsymbol{e}_j^J, \boldsymbol{e}_k^K\}$	$g(\cdot)$	$\{\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}\}$	$\{\boldsymbol{a}_i, \boldsymbol{b}_j, \boldsymbol{c}_k\}$	CP, Tucker, multiway NN, etc.	1.2.2
Representation Mapping	$\boldsymbol{\mathcal{Y}}(i, \bullet, \bullet)$, $\boldsymbol{\mathcal{Y}}(\bullet, j, \bullet)$, $\boldsymbol{\mathcal{Y}}(\bullet, \bullet, k)$		$\{\boldsymbol{M}_A, \boldsymbol{M}_B, \boldsymbol{M}_C\}$			1.2.4
Plain RNN	$\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i}$	RNN 1.13, LSTM 1.14, GRU 1.15	$\boldsymbol{W}, \boldsymbol{U}$, $\{\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{b}\}^{\{k,f,o,g\}}$, $\{\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{b}\}^{\{r,z,d\}}$	$\boldsymbol{h}_i^{[T_i]}$	LR, MHR	1.3.2
TT-RNN		TT-LSTM 1.24, TT-GRU 1.25	$\{\{\boldsymbol{\mathcal{G}}_k\}_{k=1}^d, \boldsymbol{U}, \boldsymbol{b}\}^{\{k,f,o,g\}}$, $\{\{\boldsymbol{\mathcal{G}}_k\}_{k=1}^d, \boldsymbol{U}, \boldsymbol{b}\}^{\{r,z,d\}}$		LR, AFT	1.3.4

Table 6.1: A summary of all representation learning models in this work. The first row describes our proposed framework to decompose any predictive model into a representation part and a prediction part. We have shown that a relational learning model (2nd row) is in fact an instance of the framework in the sense that the latent representations are generated by a function g of one-hot encoding vectors as input. We replace the one-hot vectors with the corresponding tensor slices to derive the representation mapping model in the 3rd row. Comparing the 4th and the 5th row one could see that we decompose the input-to-hidden weight matrix of RNNs into a train of core tensors. These RNNs generate latent representations for any succeeding prediction model.

Appendix A

A Numerical Example of User-Item Matrix Decomposition

A.1 User-Item Matrix Decomposition

A special case of KG is a user-item matrix, which forms the data basis for collaborative filtering in recommender systems [99, 35]. There exists only one type of predicate that is purchasing. Based on all items a user has purchased, the system would predict the probabilities that the user might purchase other items. It can therefore recommend to the user items that show high probabilities. This is identical to predicting the existence of an unknown fact. Here we provide a numerical example similar to [59] in Tab. A.1a, which records the purchasing of five movies by four individuals. For illustrative purpose, we design a fictive and strong pattern that the first two users tend to appreciate musical movies, while the rest of the users enjoy science fictions.

We decompose the matrix using SVD and denote this with $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$. We visualize the leading two dimensions of $\boldsymbol{U}$ and $\boldsymbol{V}$ in Fig. A.1, respectively. It is easy to see that the 2D vectors represent the users and films in a way that agrees with our designed patterns. The two users interested in science fiction are positioned closer to each other than to the users who enjoy musicals; while the representations of the movies also hint their genres.

In Tab. A.1b, we present the reconstruction of the user-item matrix with these two dimensions, namely $\hat{\boldsymbol{X}} = \boldsymbol{U}_2\boldsymbol{D}_2\boldsymbol{V}_2^T$.

The most interesting cases are where the reconstruction deviates most from the original user-item matrix: the original matrix indicates that Dominic has not yet seen Star Wars, although he is expected to be science fiction fan based on his latent representation. The

$\boldsymbol{X}$	Mary Poppins	The Sound of Music	Blade Runner	Star Wars	Alien
Amelia	1	1	0	0	1
Barbara	1	1	0	0	0
Charles	0	1	1	1	1
Dominic	0	0	1	0	1

(a) The original user-item matrix.

$\hat{\boldsymbol{X}}$	Mary Poppins	The Sound of Music	Blade Runner	Star Wars	Alien
Amelia	1.0	1.2	0.2	0.2	0.7
Barbara	1.0	0.9	-0.2	0.0	0.2
Charles	0.2	0.8	1.1	0.6	1.3
Dominic	-0.2	0.2	0.8	0.4	0.8

(b) Reconstruction of the user-item matrix with SVD $r = 2$.

Table A.1: A numerical example: a user-time matrix is a special case of a KG with one type of predicate being purchasing. A recommender system therefore corresponds to the link prediction task in context of a KG.

reconstruction, however, produces a value of 0.4 for this fact, which is for Dominic the second largest value. Therefore the recommender system would suggest Dominic to watch Star Wars, given that he has not yet done so.

In order to generalize this approach to model threefold relations, decomposition approaches of three-mode tensors have to be applied.

A.2 Derivation of Latent Representations for a New User

We continue the numerical example with the user-item matrix, in order to illustrate the advantage of learning representation mapping. We assume a new user Edward, who has only purchased Star Wars. His vector of known facts is thus $\boldsymbol{x}_{new} = (0, 0, 0, 1, 0)$. We derive the latent representation for Edward using $\boldsymbol{u}_{new}^T = \boldsymbol{x}_{new}^T \boldsymbol{V}_2 \boldsymbol{D}_2$ and plot it in the same 2D space (left plot in Fig. A.2).

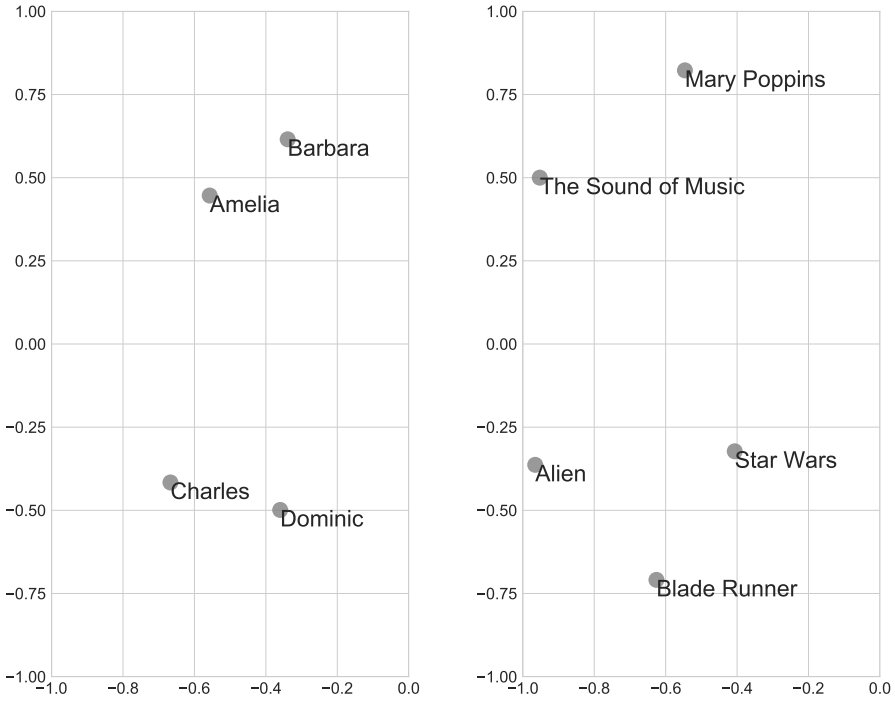


Figure A.1: Visualizing latent factors of SVD. Left: $\boldsymbol{U}_2$, the leading both factors that represent the users. Right: $\boldsymbol{V}_2$, the leading both factors that represent the movies.

We can see that based only on this one known fact, the model is already able to place Edward in a closer neighborhood with Charles and Dominic than with Amelia and Barbara. For comparison, the right plot in Fig. A.2 visualizes the latent representation derived by re-calculating the SVD. It demonstrates relatively small difference to the results without re-calculating.

We further reconstruct $\hat{\boldsymbol{x}}_{new}^T = \boldsymbol{u}_{new}^T \boldsymbol{D}_2 \boldsymbol{V}_2^T$ to show that such a model can also make rational recommendations to new users based on very few known facts.

In Tab. A.2, it is easy to see that the recommendation scores produced by mapping from known facts agree with scores by re-calculating the SVD but only with less confidence. In this numerical example, the difference in computational complexity between mapping and re-calculating is apparently insignificant. However, for larger and especially three-way tensors, retraining a decomposition matrix could be much more costly than just mapping from known facts. For simple linear models such as SVD and CP, the representation

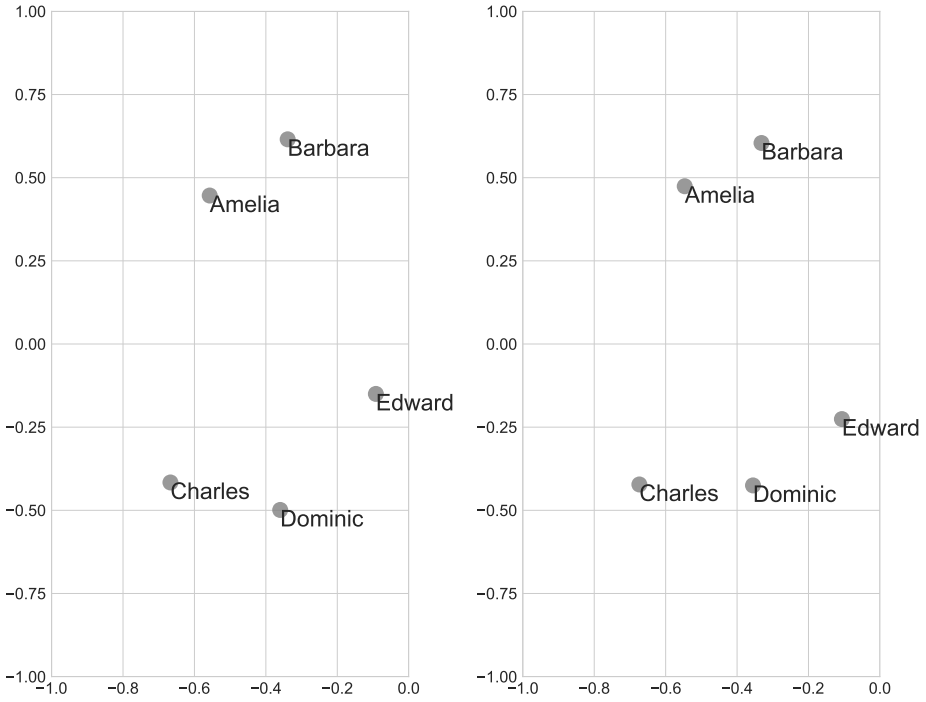


Figure A.2: Right: visualizing the latent representation of each user. The one for Edward is derived using direct mapping from $\boldsymbol{x}_{new}$. Left: Latent representations of all users that are derived by re-training the decomposition.

mapping function is free-of-charge, because it is simply the inverse of the decomposition. This is, however, not the case for non-linear models such as multi-way neural network or models that reuse representation such as RESCAL.

$\hat{\boldsymbol{X}}_{new}$	Mary Poppins	The Sound of Music	Blade Runner	Star Wars	Alien
Mapping	-0.1	0.0	0.2	0.1	0.2
Re-calculating	-0.1	0.0	0.3	0.2	0.3

Table A.2: Comparison of two reconstructions of a new user vector.

Appendix B

Forward Pass Algorithm in Tensor-Train Layer

The feed forward pass of a Tensor-Train layer can be formulated as the following algorithm.

Algorithm 1 The Calculation of a Tensor-Train Layer [85, 84]

Input: $\boldsymbol{X} \in \mathbb{R}^{S \times M}$, $(m_k)_{k=1}^d$, $(n_k)_{k=1}^d$, $(r_k)_{k=0}^d$;
Output: $\boldsymbol{Y} \in \mathbb{R}^{S \times N}$;
Initialize:
  $\{\boldsymbol{\mathcal{G}}_k\}_{k=1}^d$ with $\boldsymbol{\mathcal{G}}_k \in \mathbb{R}^{m_k \times n_k \times r_{k-1} \times r_k}$;
  $\boldsymbol{b} \in \mathbb{R}^N$;
$\boldsymbol{Z} := \boldsymbol{X}$
for $k = 1$ **to** d **do**
  $\boldsymbol{Z} := \boldsymbol{Z}.reshape(S \cdot \prod_{\substack{k'=1 \\ k' \neq k}}^{d} m_{k'},\ m_k \cdot r_{k-1})$;
  $\boldsymbol{G}_k^* := \boldsymbol{\mathcal{G}}_k.reshape(m_k \cdot r_{k-1}, r_k \cdot n_k)$;
  $\boldsymbol{Z} := \boldsymbol{Z} \cdot \boldsymbol{G}_k^*$
end for
$\boldsymbol{Y} := \boldsymbol{Z}$
for $s = 1$ **to** S **do**
  $\boldsymbol{Y}(s) = \boldsymbol{Y}(s) + \boldsymbol{b}$
end for

The scalar S is the number of data instances, i.e. the batch size. The *reshape* operator is based on the C-like row-major order.

Appendix C

Gradients in Tensor-Train Layer

Recall the forward pass of a Tensor-Train layer:

$$\widehat{\boldsymbol{\mathcal{Y}}}(j_1, j_2, ..., j_d) = \sum_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} ... \sum_{i_d=1}^{m_d} \boldsymbol{\mathcal{G}}_1(l_1) \cdot \boldsymbol{\mathcal{G}}_2(l_2) \cdot ... \cdot \boldsymbol{\mathcal{G}}_d(l_d) \cdot \boldsymbol{\mathcal{X}}(i_1, i_2, ..., i_d) + \boldsymbol{\mathcal{B}}(j_1, j_2, ..., j_d). \tag{C.1}$$

We attempt to update all necessary parameters that are responsible for mapping $(i_1, i_2, ..., i_d) \rightarrow (j_1, j_2, ..., j_d)$. The objective function is defined as the error between the output of a forward pass and the actual target:

$$E(\boldsymbol{\mathcal{Y}}(j_1, j_2, ..., j_d), \hat{\boldsymbol{\mathcal{Y}}}(j_1, j_2, ..., j_d)), \tag{C.2}$$

which is, for the sake of simplicity, shortened as E in this context.

Proof 1 (of Theorem 3)**.** *The partial derivative w.r.t a slice of an arbitrary core tensor is derived using the chain rule:*

$$\frac{\partial E}{\partial \boldsymbol{\mathcal{G}}_k(l_k)} = \frac{\partial E}{\partial \hat{\boldsymbol{\mathcal{Y}}}(j_1, j_2, ..., j_d)} \frac{\partial \hat{\boldsymbol{\mathcal{Y}}}(j_1, j_2, ..., j_d)}{\boldsymbol{\mathcal{G}}_k(l_k)} \tag{C.3}$$

For an arbitrary l_k, we apply again the chain rule on the second term on the right side:

$$\frac{\partial \hat{\boldsymbol{\mathcal{Y}}}(j_1, j_2, ..., j_d)}{\partial \boldsymbol{\mathcal{G}}_k(l_k)} = \frac{\partial \hat{\boldsymbol{\mathcal{Y}}}(j_1, j_2, ..., j_d)}{\partial \boldsymbol{H}_{d-1}(l_{d-1})} \cdot \prod_{k^*=k+1}^{d-1} \frac{\partial \boldsymbol{H}_{k^*}(l_{k^*})}{\partial \boldsymbol{H}_{k^*-1}(l_{k_*-1})} \cdot \frac{\partial \boldsymbol{H}_k(l_k)}{\partial \boldsymbol{\mathcal{G}}_k(l_k)} \tag{C.4}$$

$$= \prod_{k^*>k} \boldsymbol{\mathcal{G}}_{k^*}(l_{k^*}) \cdot \boldsymbol{\mathcal{X}}(i_1, i_2, ..., i_d) \cdot \prod_{k^*<k} \boldsymbol{\mathcal{G}}_{k^*}(l_{k^*}), \tag{C.5}$$

applying a set of ancillary variables $\{\boldsymbol{H}_k\}_{k=1}^d, \boldsymbol{H}_k \in \mathbb{R}^{(m_k \cdot n_k) \times r_k}$:

$$\boldsymbol{H}_k(l_k) = \boldsymbol{\mathcal{G}}_1(l_1) \cdot \boldsymbol{\mathcal{G}}_2(l_2) \cdot ... \cdot \boldsymbol{\mathcal{G}}_k(l_k) \cdot \boldsymbol{\mathcal{X}}(i_1, i_2, ..., i_k) \in \mathbb{R}^{r_k}. \tag{C.6}$$

Note that we apply the numerator layout convention, namely $\frac{\partial \hat{\boldsymbol{\mathcal{Y}}}(j_1,j_2,\ldots,j_d)}{\boldsymbol{\mathcal{G}}_k(l_k)} := \frac{\partial \hat{\boldsymbol{\mathcal{Y}}}(j_1,j_2,\ldots,j_d)}{\boldsymbol{\mathcal{G}}_k(l_k)^T} \in \mathbb{R}^{r_k \times r_{k-1}}$. This allows for applying the chain rule from left to right, which is more intuitive than denominator layout with right-to-left chain rule.

When, for instance, $d = 4$, the partial derivatives w.r.t. all core tensors are as follows.

$$\frac{\partial \boldsymbol{\mathcal{Y}}(j_1,j_2,j_3,j_4)}{\partial \boldsymbol{\mathcal{G}}_4(l_4)} = \boldsymbol{\mathcal{X}}(i_1,i_2,i_3,i_4) \cdot \boldsymbol{\mathcal{G}}_1(l_1) \cdot \boldsymbol{\mathcal{G}}_2(l_2) \cdot \boldsymbol{\mathcal{G}}_3(l_3) \tag{C.7}$$

$$\frac{\partial \boldsymbol{\mathcal{Y}}(j_1,j_2,j_3,j_4)}{\partial \boldsymbol{\mathcal{G}}_3(l_3)} = \frac{\partial \boldsymbol{\mathcal{Y}}(j_1,j_2,j_3,j_4)}{\partial \boldsymbol{H}_3(l_3)} \cdot \frac{\partial \boldsymbol{H}_3(l_3)}{\partial \boldsymbol{\mathcal{G}}_3(l_3)} \tag{C.8}$$

$$= \boldsymbol{\mathcal{G}}_4(l_4) \cdot \boldsymbol{\mathcal{X}}(i_1,i_2,i_3,i_4) \cdot \boldsymbol{\mathcal{G}}_1(l_1) \cdot \boldsymbol{\mathcal{G}}_2(l_2) \tag{C.9}$$

$$\frac{\partial \boldsymbol{\mathcal{Y}}(j_1,j_2,j_3,j_4)}{\partial \boldsymbol{\mathcal{G}}_2(l_2)} = \frac{\partial \boldsymbol{\mathcal{Y}}(j_1,j_2,j_3,j_4)}{\partial \boldsymbol{H}_3(l_3)} \cdot \frac{\partial \boldsymbol{H}_3(l_3)}{\partial \boldsymbol{H}_2(l_2)} \cdot \frac{\partial \boldsymbol{H}_2(l_2)}{\partial \boldsymbol{\mathcal{G}}_2(l_2)} \tag{C.10}$$

$$= \boldsymbol{\mathcal{G}}_3(l_3) \cdot \boldsymbol{\mathcal{G}}_4(l_4) \cdot \boldsymbol{\mathcal{X}}(i_1,i_2,i_3,i_4) \cdot \boldsymbol{\mathcal{G}}_1(l_1) \tag{C.11}$$

$$\frac{\partial \boldsymbol{\mathcal{Y}}(j_1,j_2,j_3,j_4)}{\partial \boldsymbol{\mathcal{G}}_1(l_1)} = \frac{\partial \boldsymbol{\mathcal{Y}}(j_1,j_2,j_3,j_4)}{\partial \boldsymbol{H}_3(l_3)} \cdot \frac{\partial \boldsymbol{H}_3(l_3)}{\partial \boldsymbol{H}_2(l_2)} \cdot \frac{\partial \boldsymbol{H}_2(l_2)}{\partial \boldsymbol{H}_1(l_1)} \cdot \frac{\partial \boldsymbol{H}_1(l_1)}{\partial \boldsymbol{\mathcal{G}}_1(l_1)} \tag{C.12}$$

$$= \boldsymbol{\mathcal{G}}_2(l_2) \cdot \boldsymbol{\mathcal{G}}_3(l_3) \cdot \boldsymbol{\mathcal{G}}_4(l_4) \cdot \boldsymbol{\mathcal{X}}(i_1,i_2,i_3,i_4) \tag{C.13}$$

Appendix D

Tensor-Train RNNs: A Simulation Study

In order to verify the implementation of Tensor-Train RNNs, we perform experiments of sequence classification. We sample MNIST [68] images to form sequences of variable lengths. If there exists one or multiple 0's in the sequence, the sequence is assigned the label of 1 and otherwise 0. This forms a standard sequence classification task that involves eventually long-term dependency, since a 0 may appear early at the beginning of the sequence. First we evaluate the classification quality on the test data set. Second, we apply Layer-wise Relevance Propagation [4, 92, 77] (LRP) to verify the classification. Obviously, if the sequence is correctly classified, then the time step that is a 0 would have a significant positive contribution to the classification in term of relevance scores. In Tab. D.1 we report the average and standard deviation of 5 runs of experiments. In each run of the experiment, we generate 10,000 sequences for training and 1,000 for test. A sequence has at maximum 32 MNIST digits. The plain LSTM model has a hidden size of size 64. The Tensor-Train LSTM factorizes the input dimension of 28×28 as $7 \times 7 \times 16$, and the output dimension as $4 \times 4 \times 4$. The TT ranks are $[1, 4, 4, 1]$. We evaluate the prediction of both models using accuracy score. Since the relevance scores may range in $\mathbb{R}$, we use AUROC as evaluation metric. For the Tensor-Train layer in the TT-LSTM, we reconstruct the full weight matrix $\boldsymbol{W}$ from the core tensors applying Eq. 1.16 to calculate the relevance scores.

One could draw the conclusion from Tab. D.1 that the TT-LSTM performs equally well in term of accuracy, requiring only approximately 0.5% of weight parameters as in plain LSTM. However, the relevance scores have become slightly worse. In Fig. D.1 we visualize three random sequences that are correctly classified.

Model	Prediction accuracy	LRP AUROC	No. weights
LSTM	0.972 ± 0.001	0.987 ± 0.003	200,704
TT-LSTM	0.970 ± 0.006	0.955 ± 0.017	1,152

Table D.1: Evaluation of prediction and relevance recognition by LSTM and TT-LSTM on MNIST sequence data.

One could see here that the relevance scores of TT-LSTM are less stable than those of plain LSTM, probably due to the fact that the weights are in fact reconstruction from the core tensors. In the first two sequences one could see that both models can store the deciding pattern for the classification task over a long period of time, since the zeros at the beginning of both sequences have received high relevance scores. Another interesting fact one would notice is that the 8th digit in the third sequence is mistakenly recognized by both LSTM and TT-LSTM as a 0, although the digit is a 6 that looks like a 0. This is a empirical proof that both models are capable of identifying the positive correlation between a circle-like pattern in the sequence and the label of the sequence, only by processing enough training data that define this correlation.

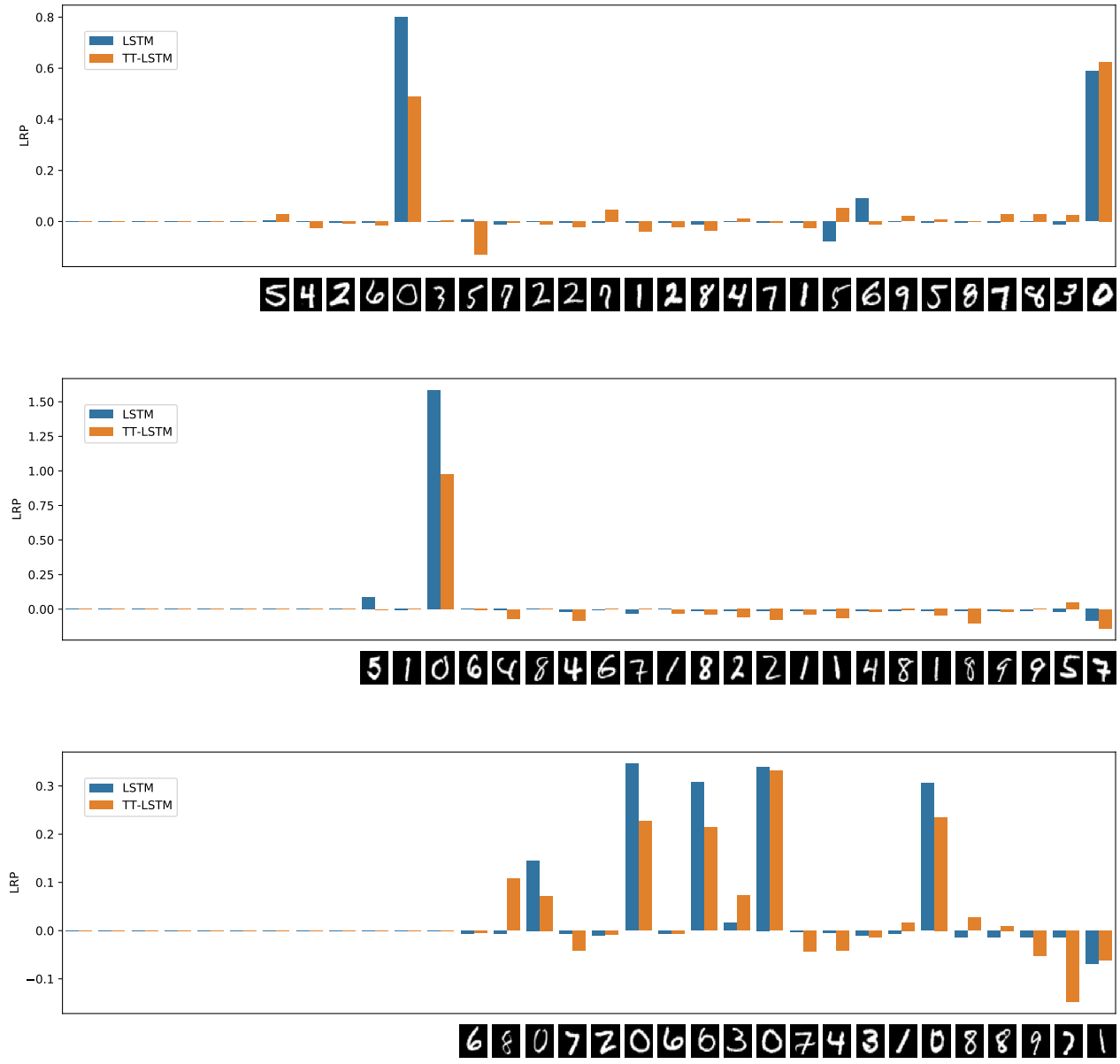


Figure D.1: Relevance scores calculated with LRP for each digit in artificial sequences.

List of Figures

List of Tables

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Marc Antonini, Michel Barlaud, Pierre Mathieu, and Ingrid Daubechies. Image coding using wavelet transform. *IEEE Transactions on image processing*, 1(2):205–220, 1992.

[3] Relja Arandjelović and Andrew Zisserman. Look, listen and learn. *arXiv preprint arXiv:1705.08168*, 2017.

[4] Leila Arras, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. Explaining recurrent neural network predictions in sentiment analysis. *arXiv preprint arXiv:1706.07206*, 2017.

[5] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. *The semantic web*, pages 722–735, 2007.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[7] Bram Bakker. Reinforcement learning with long short-term memory. In *Advances in neural information processing systems*, pages 1475–1482, 2002.

[8] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[9] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[10] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[11] Yoshua Bengio, Ian J Goodfellow, and Aaron Courville. Deep learning. *Nature*, 521:436–444, 2015.

[12] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[13] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[14] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM, 2008.

[15] Roger D Boyle and Richard C Thomas. *Computer vision: A first course*. Blackwell Scientific Publications, Ltd., 1988.

[16] Leo Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001.

[17] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

[18] Thomas Brox and Jitendra Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE transactions on pattern analysis and machine intelligence*, 33(3):500–513, 2011.

[19] R Caruna. Multitask learning: A knowledge-based source of inductive bias. In *Machine Learning: Proceedings of the Tenth International Conference*, pages 41–48, 1993.

[20] Eva Ceulemans and Iven Van Mechelen. Tucker2 hierarchical classes analysis. *Psychometrika*, 69(3):375–399, 2004.

[21] Devendra K Chaturvedi. *Soft computing: techniques and its applications in electrical engineering*, volume 103. Springer, 2008.

[22] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[23] Jie-Zhi Cheng, Dong Ni, Yi-Hong Chou, Jing Qin, Chui-Mei Tiu, Yeun-Chung Chang, Chiun-Sheng Huang, Dinggang Shen, and Chung-Ming Chen. Computer-aided diagnosis with deep learning architecture: applications to breast lesions in us images and pulmonary nodules in ct scans. *Scientific reports*, 6:24454, 2016.

[24] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[25] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[26] Edward Choi, Mohammad Taha Bahadori, Andy Schuetz, Walter F Stewart, and Jimeng Sun. Doctor ai: Predicting clinical events via recurrent neural networks. In *Machine Learning for Healthcare Conference*, pages 301–318, 2016.

[27] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[28] Dan CireşAn, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012.

[29] Daniel Crevier. *AI: The tumultuous history of the search for artificial intelligence*. Basic Books, 1993.

[30] Balázs Csanád Csáji. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24:48, 2001.

[31] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[32] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[33] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[34] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610. ACM, 2014.

[35] Michael D Ekstrand, John T Riedl, Joseph A Konstan, et al. Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, 4(2):81–173, 2011.

[36] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[37] Cristóbal Esteban, Danilo Schmidt, Denis Krompaß, and Volker Tresp. Predicting sequences of clinical events by using a personalized temporal latent embedding model. In *Healthcare Informatics (ICHI), 2015 International Conference on*, pages 130–139. IEEE, 2015.

[38] Cristóbal Esteban, Oliver Staeck, Stephan Baier, Yinchong Yang, and Volker Tresp. Predicting clinical events by combining static and dynamic information using recurrent neural networks. In *Healthcare Informatics (ICHI), 2016 IEEE International Conference on*, pages 93–101. IEEE, 2016.

[39] Zeno Gantner, Lucas Drumond, Christoph Freudenthaler, Steffen Rendle, and Lars Schmidt-Thieme. Learning attribute-to-feature mappings for cold-start recommendations. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 176–185. IEEE, 2010.

[40] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[41] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[42] Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237. ACM, 1999.

[43] G Hinton, N Srivastava, and K Swersky. Rmsprop: Divide the gradient by a running average of its recent magnitude. *Neural networks for machine learning, Coursera lecture 6e*, 2012.

[44] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[45] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91, 1991.

[46] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[47] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[48] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[49] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[50] JNR Jeffers. Two case studies in the application of principal component analysis. *Applied Statistics*, pages 225–236, 1967.

[51] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[52] MG Kendall. A course in multivariate statistics, 1957.

[53] Henk AL Kiers. Towards a standardized notation and terminology in multiway analysis. *Journal of chemometrics*, 14(3):105–122, 2000.

[54] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *AAAI*, pages 2741–2749, 2016.

[55] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[56] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*, 2017.

[57] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE, 1995.

[58] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[59] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

[60] Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1863–1871, Bejing, China, 22–24 Jun 2014. PMLR.

[61] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[62] Denis Krompaß. Exploiting prior knowledge and latent variable representations for the statistical modeling and probabilistic querying of large knowledge graphs. November 2015.

[63] Denis Krompaß, Stephan Baier, and Volker Tresp. Type-constrained representation learning in knowledge graphs. In *International Semantic Web Conference*, pages 640–655. Springer, 2015.

[64] Denis Krompaß, Cristóbal Esteban, Volker Tresp, Martin Sedlmayr, and Thomas Ganslandt. Exploiting latent embeddings of nominal clinical data for predicting hospital readmission. *KI-Künstliche Intelligenz*, 29(2):153–159, 2015.

[65] Ray Kurzweil, Robert Richter, Ray Kurzweil, and Martin L Schneider. *The age of intelligent machines*, volume 579. MIT press Cambridge, 1990.

[66] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10(Jan):1–40, 2009.

[67] Ora Lassila, Ralph R Swick, et al. Resource description framework (rdf) model and syntax specification. 1998.

[68] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[69] Philip Leith. The rise and fall of the legal expert system. *European Journal of Law and Technology*, 1(1):179–201, 2010.

[70] Daniel Lévy and Arzav Jain. Breast mass classification from mammograms using deep convolutional neural networks. *arXiv preprint arXiv:1612.00542*, 2016.

[71] Alexandra L'Heureux, Katarina Grolinger, Hany F ElYamany, and Miriam Capretz. Machine learning with big data: Challenges and approaches. *IEEE Access*, 2017.

[72] Pyry Matikainen, Martial Hebert, and Rahul Sukthankar. Trajectons: Action recognition through the motion analysis of tracked features. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 514–521. IEEE, 2009.

[73] Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *Interspeech*, pages 3771–3775, 2013.

[74] Tomas Mikolov. *Statistical Language Models Based Neural Networks*. PhD thesis, Brno University of Technology, 2012.

[75] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[76] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *hlt-Naacl*, volume 13, pages 746–751, 2013.

[77] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, pages 211–222, 2017.

[78] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L Cun. Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pages 739–746, 2006.

[79] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[80] Allen Newell and Herbert A Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19(3):113–126, 1976.

[81] Maximilian Nickel. Tensor factorization for relational learning. August 2013.

[82] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

[83] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 809–816, 2011.

[84] Alexander Novikov. Tensornet. `https://github.com/Bihaqo/TensorNet`, 2015.

[85] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pages 442–450, 2015.

[86] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[87] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[88] Paulo E Rauber, Samuel G Fadel, Alexandre X Falcao, and Alexandru C Telea. Visualizing the hidden activity of artificial neural networks. *IEEE transactions on visualization and computer graphics*, 23(1):101–110, 2017.

[89] Elaine Rich and Kevin Knight. Artificial intelligence. *McGraw-Hill, New*, 1991.

[90] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[91] S Russell. Artificial intelligence: A modern approach author: Stuart russell, peter norvig, publisher: Prentice hall pa. 2009.

[92] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[93] Shikhar Sharma, Ryan Kiros, and Ruslan Salakhutdinov. Action recognition using visual attention. *arXiv preprint arXiv:1511.04119*, 2015.

[94] Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150, 1995.

[95] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[96] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[97] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[98] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[99] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.

[100] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.

[101] Deqing Sun, Stefan Roth, and Michael J Black. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *International Journal of Computer Vision*, 106(2):115–137, 2014.

[102] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[103] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[104] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, 2016.

[105] Gregory Michael Thorson, Christopher Aaron Clark, and Dan Luu. Vector computation unit in a neural network processor, December 22 2016. US Patent App. 15/389,288.

[106] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. Compressing recurrent neural network with tensor train. *arXiv preprint arXiv:1705.08052*, 2017.

[107] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[108] Gerhard Tutz. *Regression for categorical data*, volume 34. Cambridge University Press, 2011.

[109] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[110] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[111] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Action recognition by dense trajectories. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3169–3176. IEEE, 2011.

[112] Jinhua Wang, Xi Yang, Hongmin Cai, Wanchang Tan, Cangzheng Jin, and Li Li. Discrimination of breast cancer with microcalcifications on mammography by deep learning. *Scientific reports*, 6:27327, 2016.

[113] Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Recurrent policy gradients. *Logic Journal of the IGPL*, 18(5):620–634, 2010.

[114] Yinchong Yang, Cristóbal Esteban, and Volker Tresp. Embedding mapping approaches for tensor factorization and knowledge graph modelling. In Harald Sack, Eva Blomqvist, Mathieu d'Aquin, Chiara Ghidini, Simone Paolo Ponzetto, and Christoph Lange, editors, *The Semantic Web. Latest Advances and New Domains: 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 – June 2, 2016, Proceedings*, pages 199–213. Springer International Publishing, 2016.

[115] Yinchong Yang, Peter A. Fasching, and Volker Tresp. Modeling progression free survival in breast cancer with tensorized recurrent neural networks and accelerated failure time model. In *Machine Learning for Healthcare 2017*, volume 68 of *Proceedings of Machine Learning Research*, Northeastern University, Boston, USA, 18–19 Aug 2017. JMLR.

[116] Yinchong Yang, Peter A. Fasching, and Volker Tresp. Predictive modeling of therapy decisions in metastatic breast cancer with recurrent neural network encoder and multinomial

hierarchical regression decoder. In *Proceedings of the IEEE International Conference on Healthcare Informatics (ICHI)*, Park City, Utah, USA, 23–26 Aug 2017. IEEE.

[117] Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3891–3900, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. JMLR.

[118] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[119] Jing Zhao. Temporal weighting of clinical events in electronic health records for pharmacovigilance. In *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on*, pages 375–381. IEEE, 2015.