Learning Representations for Supervised Information Fusion using Tensor Decompositions and Deep Learning Methods

Stephan Baier

München 2018

Learning Representations for Supervised Information Fusion using Tensor Decompositions and Deep Learning Methods

Stephan Baier

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

vorgelegt von
Stephan Baier
aus Eggenfelden

München, den 23.10.2018

Erstgutachter: Prof. Dr. Volker Tresp

Zweitgutachter: Prof. Dr. Dr. Lars Schmidt-Thieme

Drittgutachter: Prof. Dr. Morten Mørup

Tag der mündlichen Prüfung: 27.03.2019

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.

Baier, Stephan

Name, Vorname

München, 23.10.2018

Ort, Datum

Stephan Baier

Unterschrift Doktorand/in

Formular 3.2

Contents

List of Figures

List of Tables

Abstract

Machine learning is aimed at the automatic extraction of semantic-level information from potentially raw and unstructured data. A key challenge in building intelligent systems lies in the ability to extract and fuse information from multiple sources. In the present thesis, this challenge is addressed by using representation learning, which has been one of the most important innovations in machine learning in the last decade. Representation learning is the basis for modern approaches to natural language processing and artificial neural networks, in particular deep learning, which includes popular models such as convolutional neural networks (CNN) and recurrent neural networks (RNN). It has also been shown that many approaches to tensor decomposition and multi-way models can also be related to representation learning. Tensor decompositions have been applied to a variety of tasks, e.g., knowledge graph modeling and electroencephalography (EEG) data analysis. In this thesis, we focus on machine learning models based on recent representation learning techniques, which can combine information from multiple channels by exploiting their inherent multi-channel data structure.

This thesis is divided into three main sections. In the first section, we describe a neural network architecture for fusing multi-channel representations. Additionally, we propose a self-attention mechanism that dynamically weights learned representations from various channels based on the system context. We apply this method to the modeling of distributed sensor networks and demonstrate the effectiveness of our model on three real-world sensor network datasets.

In the second section, we examine how tensor factorization models can be applied to modeling relationships between multiple input channels. We apply tensor decomposition models, such as CANDECOMP/PARAFAC (CP) and tensor train decomposition, in a novel way to high-dimensional and sparse data tensors, in

addition to showing how they can be used for machine learning tasks, such as regression and classification. Furthermore, we illustrate how the tensor models can be extended to continuous inputs by learning a mapping from the continuous inputs to the latent representations. We apply our approach to the modeling of inverse dynamics, which is crucial for accurate feedforward robot control. Our experimental results show competitive performance of the proposed functional tensor model, with significantly decreased training and inference time when compared to state-of-the-art methods.

In the third part, we show how the multi-modal information from both a statistical semantic model and a visual model can be fused to improve the task of visual relationship detection. In this sense, we combine standard visual models for object detection, based on convolutional neural networks, with latent variable models based on tensor factorization for link prediction. Specifically, we propose two approaches for the fusion of semantic and sensory information. The first approach uses a probabilistic framework, whereas the second makes use of a multi-way neural network architecture. Our experimental results on the recently published Stanford Visual Relationship dataset, a challenging real-world dataset, show that the integration of a statistical semantic model using link prediction methods can significantly improve visual relationship detection.

Zusammenfassung

Maschinelles Lernen zielt auf die automatische Extraktion semantischer Information aus zum Teil rohen und unstrukturierten Daten. Eine entscheidende Herausforderung beim Entwurf intelligenter Systeme, besteht darin Informationen aus verschiedenen Quellen zu extrahieren und zu fusionieren. In dieser Arbeit wird diesen Herausforderungen mit Methoden des Repräsentations-Lernens begegnet, welche eine der bedeutendsten Innovationen im Maschinellen Lernen in der letzten Dekade darstellt. Repräsentations-Lernen ist die Basis für moderne Ansätze zur Verarbeitung natürlicher Sprache und Modellierung künstlicher Neuronaler Netze, insbesondere dem Deep Learning, welchem beliebte Modelle wie Convolutional Neural Networks (CNN) und rekurrente neuronale Netze (RNN) zugeordnet werden. Außerdem wurde gezeigt, dass auch viele Ansätze zur Tensor Faktorisierung und Multi-way Modelle als Repräsentations-Lernen interpretiert werden können. Tensor Faktorisierungs Modelle finden Anwendung in einer Vielzahl von Bereichen, wie zum Beispiel der Modellierung von Wissensgraphen und der Elektroenzephalografie (EEG) Daten Analyse. Die hier vorliegende Arbeit konzentriert sich auf aktuelle Techniken des Repräsentations-Lernens, welche Information aus unterschiedlichen Kanälen kombinieren und dabei die inhärente Mehr-Kanal Struktur der Daten ausnutzen.

Die Arbeit ist in drei Hauptteile gegliedert. Im ersten Teil wird die Architektur eines neuronalen Netzes beschrieben, welches zur Fusion mehrerer Repräsentationen aus unterschiedlichen Kanälen verwendet wird. Des Weiteren wird ein Attention Mechanismus vorgestellt, welcher dynamisch die gelernten Repräsentationen aus unterschiedlichen Kanälen in Abhängigkeit des aktuellen Systemzustands gewichtet. Die Methode wird zur Modellierung verteilter Sensor Netzwerke angewendet. Dabei wird die Effektivität des Ansatzes anhand dreier Datensätze mit echten Sensor

Werten evaluiert.

Im zweiten Teil dieser Arbeit wird untersucht, wie Tensor-Faktorisierungs Modelle zur Modellierung von Beziehungen zwischen verschiedenen Eingangs Kanälen verwendet werden können. Dabei werden Tensor Modelle wie CANDECOMP/PARAFAC (CP) und Tensor Train in einer neuartigen Art und Weise auf hochdimensionale und dünnbesetzte Tensoren angewendet. Es wird gezeigt, wie diese Modelle für Aufgaben des maschinellen Lernens, wie Regression und Klassifikation eingesetzt werden können. Desweitern wird gezeigt, wie die Tensor Modelle zu kontinuierlichen Eingangsvariablen erweitert werden können, indem eine Funktion von den kontinuierlichen Eingängen zu der latenten Repräsentation des Faktorisierungs Modells gelernt wird. Der gezeigte Ansatz wird schließlich zur Modellierung inverser Dynamiken angewandt. Die Modellierung inverser Dynamiken ist essenziell für die Vorwärtssteuerung eines Roboters. Die Experimente zeigen, dass das kontinuierliche Tensor Modell vergleichbare Ergebnisse erzielt wie herkömmliche Methoden für diese Aufgabe, wobei sich durch das Tensor Modell sowohl die Trainings als auch die Inferenz Zeit deutlich reduzieren lassen.

Im dritten Teil wird gezeigt, wie die multi-modale Information eines statistisch semantischen Modells und eines visuellen Modells fusioniert werden können, um im Bereich der visuellen Infromationsextraktion, speziell dem Erkennen von Beziehungen zwischen visuellen Objekten, verbesserte Ergebnisse zu erzielen. Dabei wird ein gängiges, auf CNNs basierendes, visuelles Modell zur Objekterkennung mit Tensor-Faktorisierungs Modellen zur Modellierung von Wissensgraphen kombiniert. Es werden zwei Ansätze für die Fusion semantischer und sensorischer Information gezeigt. Der erste Ansatz benutzt eine probabilistische Methode, wohingegen der zweite Ansatz ein Multi-way neuronales Netzwerk verwendet um die Informationen zu kombinieren. Die Evaluation auf einem kürzlich veröffentlichten Datensatz (Stanford Visual Relationship Dataset), mit Bildern aus der realen Welt, zeigt, dass die Integration eines statistisch semantischen Modells, die Methoden zur Detektion visueller Objektbeziehungen deutlich verbessert.

Acknowledgement

This dissertation is based on the research work that I carried out as a Ph.D. student in a joint program between the University of Munich (LMU) and the Machine Intelligence Group at Siemens AG. During that time, I have been extremely fortunate to be surrounded by inspiring people who helped me develop both academically and personally.

Firstly, I would like to express my sincere gratitude to my advisor Prof. Dr. Volker Tresp for the continuous support of my Ph.D. study and related research. I could not have imagined having a better advisor and mentor for my Ph.D. study. During the course of my Ph.D., he constantly provided me with new perspectives on the research field. With his enormous experience, he always gave me valuable pieces of advice and guided me in the right direction. Many thanks to Volker for the trust he gives to his students, and for the many extended hours of discussions on various topics around machine learning. Special thanks also goes to Prof. Dr. Dr. Lars Schmidt-Thieme for agreeing to be the second examiner of my thesis.

Many thanks to all the great colleagues and fellow Ph.D. students at Siemens, especially Dr. Yinchong Yang, Cristóbal Esteban, Dr. Denis Krompass, Sanjeev Kumar Karn, Yunpu Ma, Marcel Hildebrandt, and Dr. Sigurd Spieckermann for long hours of discussion on machine learning topics as well as the wonderful collaboration on joint research papers. I would like to also thank the journal club community for great talks and enthusiastic discussions on latest deep learning research. I further thank Yi Huang, who guided me during my first days at Siemens as a working student and Dr. Ulli Waltinger for continuously supporting Ph.D. research at Siemens.

Greatest thanks I owe to my family for always supporting and helping me: My brother Christoph, for being a great companion, and my parents Anna and

Karl, for their inexhaustible support, in all aspects. My deepest thanks go to my wonderful wife Annika, for her love, her patience, and for allowing me to freely pursue my research during all the years. Thank you very much for always being there for me and making me a happy person.

Chapter 1

Introduction

In this chapter, we introduce the technical background of the thesis. In Section 1.1, we present the mathematical notation, that we use throughout the work. In Section 1.2, we introduce the field of representation learning, a sub-area of machine learning research, which has gained notable attention in the last decade. In traditional machine learning, often sophisticated features are extracted from the raw data in a preprocessing step before a predictive model is fit. The goal of representation learning is to learn the best features for the prediction task directly from the data instead of hand crafting them. In Section 1.3, we introduce and discuss information fusion. The ability to fuse information into a robust percept is a central aspect of many intelligent systems. The human brain constantly integrates information derived from various senses and from semantic and episodic memory. Technical applications of information fusion can be found, for example, in distributed sensor networks, where information needs to be integrated from multiple channels, or in visual scene description tasks, where sensory data needs to be modeled together with semantic information. There is a close relationship between machine learning and information fusion research, as both are concerned with the process of turning raw data into semantic-level decisions. In this work, we consider the modeling of the information fusion process using supervised machine learning techniques.

1.1 Notation

In the following, scalars will be denoted by lowercase letters x and by uppercase letters X if they represent constants. Column vectors will be denoted by bold lowercase letters $\boldsymbol{x}$, and matrices will be denoted by bold uppercase letters $\boldsymbol{X}$. Tensors of order three or higher will be denoted by bold calligraphic uppercase letters $\boldsymbol{\mathcal{X}}$.

The i-th index of a vector will be denoted as $\boldsymbol{x}(i)$, the element at row i and column j of a matrix will be denoted as $\boldsymbol{X}(i, j)$. Elements of higher order tensors indexed by i_1 to i_n are denoted in a similar way as $\boldsymbol{\mathcal{X}}(i_1, \ldots, i_n)$. Subtensors will be denoted by using a colon instead of an index. For example $\boldsymbol{\mathcal{X}}(i, :, :)$ denotes the matrix, sliced at position i on the first mode of the tensor $\boldsymbol{\mathcal{X}}$. Further notations are defined throughout the work as needed.

1.2 Introduction to Representation Learning

Neural representations play an important role in the human brain for many cognitive functions, such as perception, memory, decision making, and motor control. Motivated by the biology, neural like representations also have a long tradition in artificial intelligence research. Especially, with the rise of deep learning, the learning of expressive representations has become a main concern in machine learning. In Section 1.2.1, we give an overview of learning representations. In Sections 1.2.2 and 1.2.3, respectively, we introduce two widely used methods for learning representations, namely neural networks and tensor decompositions.

1.2.1 Learning Representations

Within the last decade, representation learning has gained tremendous attention in the machine learning community. Modeling complex structured data, such as images, sensory data, or text, requires a transformation of the raw input data into a representation space, in which modeling becomes feasible. Traditionally, this transformation has been achieved by extracting data-specific features, often defined by human experts, in a preprocessing step before training the machine

learning model. For example, to derive a feature representation of an image, expert-designed features such as SIFT [90] and HOG [39], which are based on key-point detection and hand-crafted features such as color gradients, have been extracted. In representation learning, the feature extraction becomes part of the machine learning model, instead of being delegated to a preprocessing step. The parameters of a differentiable mapping function, which maps from the raw input space, e.g., pixels, to a latent representation space, are learned from training data. The development in representation learning has mainly been driven by advances in the field of artificial neural networks, often referred to as deep learning. According to Goodfellow et al. [62], the concept of representation learning ties together all the many forms of deep learning. Neural networks are universal function approximators, which can have various architectures. Popular instances are convolutional neural networks (CNN) and recurrent neural networks (RNN). These methods are suitable for learning a mapping function for unstructured sensory data such as images or speech, and also for learning representations of discrete entities. For discrete entities, such as words or graph nodes, the latent representation can be directly learned by the machine learning model without explicitly learning a mapping function. These latent representations are often referred to as embeddings. Besides neural networks, embeddings are also often learned using factorization techniques, such as matrix or tensor factorization.

Representation learning can be conducted in a supervised or unsupervised manner. In unsupervised learning, no labeled data is available, and the model typically tries to estimate $p(\boldsymbol{x}|\boldsymbol{h})$, where $\boldsymbol{x}$ is the data and $\boldsymbol{h}$ is the latent representation. The learned features in $\boldsymbol{h}$ should correspond to the latent explanatory factors, which are causing $\boldsymbol{x}$ [70, 62]. If $\boldsymbol{h}$ is learned in an unsupervised fashion, the representation is only useful for a predictive model $p(\boldsymbol{y}|\boldsymbol{h})$ if $\boldsymbol{y}$ depends on the latent causes among $\boldsymbol{h}$. Learning $\boldsymbol{h}$ in a predictive model $p(\boldsymbol{y}|\boldsymbol{x})$ forces the model to encode the relevant factors for predicting $\boldsymbol{y}$. In supervised learning, the input $\boldsymbol{x}$ is mapped to a latent representation $\boldsymbol{h}$, and from that mapping, a label $\boldsymbol{y}$ is predicted. The model is trained end-to-end so that the latent representation $\boldsymbol{h}$ is derived as a by-product of learning $p(\boldsymbol{y}|\boldsymbol{x})$. For example, latent representations for images can be derived from an image classification model, typically implemented as a CNN, where the latent representation $\boldsymbol{h}$, describing the image is the output

of the second last network layer.

Bengio et al. [18] give the following motivation for using vector representations: If we consider a representation vector with n features and k values, k^n distinct concepts can be represented. If we want to represent more than n concepts, the representations necessarily need to share some features, thereby enabling generalization. Having shared features, the vector representations define a meaningful similarity space, where semantically similar entities cluster together. This space distinguishes the learned representations from symbolic representations, e.g., a one-hot representation, where all entities have the same distance among each other. Reducing the representation vector's dimensionality, is one method of regularization. Further regularization can be achieved by introducing priors such as sparseness, smoothness, disentanglement, and simplicity of factor dependencies [18].

Representation learning is useful for many tasks in machine learning. Greedy layer-wise unsupervised pre-training of deep believe networks started the revival of neural network research, in 2006 [69]. Representation learning can be used for a number of transfer tasks, such as clustering, outlier detection, one-shot, and zero-shot learning. Furthermore, interest is increasing in explainable artificial intelligence. One possible way of achieving this is by trying to find interpretable structures in the learned latent representations. In Section 1.3.3, we will discuss how the concept of representation learning is also useful for model-based information fusion. Recent surveys on representation learning can be found in [18] and in Chapter 15 of [62].

1.2.2 Neural Networks

Artificial neural networks originate from neuroscience as a mathematical model for the dynamics of neurons in the brain [95, 123]. In machine learning research, neural networks have become popular as universal function approximators for pattern recognition. In the last decade, neural networks have received tremendous attention often using the term deep learning [62]. It has been shown that only one hidden layer, with sufficiently many hidden units, is enough to approximate any Borel measurable function up to any desired non-zero precision [75]. Thus,

neural networks are considered to be universal function approximators. However, in practice the hidden layer might need to be infeasible large, and the learning algorithm might fail to find the correct parameter setting. It has been shown that for some families of functions deep models with many hidden layers can overcome these problems [93, 19, 69, 62]. Another motivation for deep network architectures arises from the view of representation learning. It is assumed that every layer in the neural network extracts a representation which is formed of underlying exploratory factors of its input. For complex structured data, such as images, it is reasonable to assume that these factors are hierarchically structured. This gives rise to the use of multiple successive layers. Various types of neural network architectures have been proposed. In the following, we review the most important architectures, which are extensively used in recent neural network research.

Multilayer Perceptrons Multilayer perceptrons (MLP) are the most basic notion of a neural network. They are simple feed-forward neural networks without circles. Each layer of a MLP is a function which transforms an input vector to an output vector. Arbitrary many layers are applied successively. The parameters in a hidden layer $l \in [1, \ldots, L]$ are the weight matrix $\boldsymbol{W}^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ and the bias vector $\mathbf{b}^{(l)} \in \mathbb{R}^{d_l}$. The transformation is computed as

$$\boldsymbol{h}^{(l)} = \psi(\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}); \qquad \boldsymbol{h}^{(0)} = \boldsymbol{x}, \tag{1.1}$$

where $\boldsymbol{x} \in \mathbb{R}^{d_0}$ is the input vector to the network, and ψ is a non-linear activation function, such as hyperbolic tangent (tanh) or the logistic sigmoid, which is applied element-wise to the vector. The output of the last hidden layer $\boldsymbol{h}^{(L)}$ is then passed to the output layer, which calculates the output $\hat{\boldsymbol{y}}$, as

$$\hat{\boldsymbol{y}} = \phi(\boldsymbol{A}\boldsymbol{h}^{(L)} + \boldsymbol{c}), \tag{1.2}$$

where $\boldsymbol{A} \in \mathbb{R}^{k \times d_L}$ is the weight matrix and $\boldsymbol{c} \in \mathbb{R}^k$ is the bias vector of the output layer, with $k \in \mathbb{N}$ denoting the output dimensionality. ϕ is an activation function, which relates the model output to the parameter of the distribution of $\hat{\boldsymbol{y}}$. For example if $\hat{\boldsymbol{y}}$ is a Bernoulli variable, ϕ is the logistic sigmoid, and if $\hat{\boldsymbol{y}}$ is a categorical variable, ϕ is the softmax function.

Convolutional Neural Networks Since the great success of a model called AlexNet [84] in the Image Net Challange 2012 [125] convolutional neural networks (CNN) are ubiquitous in machine learning. They became the state-of-the-art method for many image related machine learning tasks. CNNs have also been applied to other tasks involving non image data, such as text. One of the first methods with a structure similar to today's models can be found in [87]. The architecture of CNNs is inspired by the visual cortex. The neurons in the visual cortex only respond to a limited area in the visual field which is called the receptive field. By using the convolution operation the weights in a CNN are also applied to local image patches.

In a convolutional layer, the input is convolved by a parametrized kernel. The transformation is computed as

$$\boldsymbol{h}_j^{(l)} = \psi\Big(\sum_{i=1}^{K^{(l-1)}} \boldsymbol{w}_{ij} * \boldsymbol{h}_i^{(l-1)} + \boldsymbol{b}_j^{(l)}\Big); \qquad \boldsymbol{h}^{(0)} = \boldsymbol{x} \tag{1.3}$$

where ψ is a non-linear activation function, $\boldsymbol{b}$ is the bias term, $\boldsymbol{w}$ is the convolution kernel and $*$ denotes the convolution operation. At each layer, multiple outputs $\boldsymbol{h}_j^{(l)}$ with $j \in \left[1 \dots K^{(l)}\right]$ are computed. All outputs are then input to the next layer. The intermediate outputs of convolutional layers are called feature maps. The convolution operation can also be applied in multiple dimensions, e.g., two dimensional convolution on images, or three dimensional convolution on videos. If the input is an image, all vectors $\boldsymbol{h}_i$, $\boldsymbol{w}_{ij}$, and $\boldsymbol{b}_j$ are replaced by matrices, and the initial feature maps are typically the three color channels of the input image ($K^{(0)} = 3$).

The convolutional layer can be interpreted as a fully connected layer where some of the weights are shared through the convolution operation. In CNN architectures the convolutional layer is often followed by a pooling layer. Pooling layers reduce the dimensionality of each feature map by down sampling. CNNs are networks which make use of convolutional layers. Most popular CNN architectures, such as LeNet or AlexNet, apply multiple convolutional layers, each followed by a pooling layer. The output of the last convolution layer is reshaped into a vector which is then mapped to the output, either by a linear unit or a MLP. CNN architectures

have been extended in many ways. Among the state-of-the-art architectures are VGG [132], ResNet [68], and Inception network [141].

Recurrent Neural Networks Recurrent neural networks (RNN) are designed for modeling sequential data. They have shown state-of-the-art performance in many important machine learning applications, such as automatic speech recognition, end-to-end machine translation, and image caption generation. Given an input sequence $x = (\boldsymbol{x}_1, ..., \boldsymbol{x}_T)$ of length $T \in \mathbb{N}$ with $\boldsymbol{x}_t \in \mathbb{R}^n$, an RNN generates a hidden state sequence $h = (\boldsymbol{h}_1, ..., \boldsymbol{h}_T)$, $\boldsymbol{h}_t \in \mathbb{R}^d$, where the hidden representation $\boldsymbol{h}_t$ at time $t \in \{1, 2, ..., T\}$ depends on the previous hidden state $\boldsymbol{h}_{t-1}$ and the current input $\boldsymbol{x}_t$ as

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t). \tag{1.4}$$

At every time step an output $\hat{\boldsymbol{y}}_t \in \mathbb{R}^k$, may be computed given the current hidden state $\boldsymbol{h}_t$ as

$$\hat{\boldsymbol{y}}_t = g(\boldsymbol{h}_t). \tag{1.5}$$

Most commonly f is an affine transformation, parameterized by $\boldsymbol{U} \in \mathbb{R}^{d \times d}$, $\boldsymbol{W} \in \mathbb{R}^{d \times n}$, and $\boldsymbol{b} \in \mathbb{R}^d$, followed by an element-wise non-linear activation function, e.g. tanh, so that

$$\boldsymbol{h}_t = \psi(\boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}). \tag{1.6}$$

In regression tasks, g is typically chosen to be a simple affine transformation and in multi-class classification tasks g is usually chosen to be an affine transformation followed by the softmax function. It has been shown that the basic RNN suffers from the problem of vanishing gradient for long input sequences in the training data [20, 72]. Therefore, more sophisticated components like the long short-term memory units (LSTM) [73] and the gated recurrent units (GRU) [33, 36] have been proposed. Various methods for training RNNs have been explored. However, in most recent applications backpropagation through time has been used. As RNNs process the input sequence x iteratively (see eq. 1.4), they can run over a data stream, which generates x, and compute the latest hidden state without having to store past measurements. This property makes them suitable for stream processing.

Encoder-Decoder Framework The encoder-decoder framework is a general architecture mapping an input structure to an output structure using artificial neural networks. They consist of an encoder and a decoder part, which can be modeled by any neural network architecture, e.g., MLP, CNN, or RNN. The encoder function f_{enc} is trained to produce a latent representation of its input data being, e.g., an image or a sentence, such that

$$\boldsymbol{c} = f_{\text{enc}}(\boldsymbol{x}). \tag{1.7}$$

The latent representation $\boldsymbol{c} \in \mathbb{R}^d$, is then passed to another neural network f_{dec}, which represents the decoder function. The decoder produces an output given the latent representation $\boldsymbol{c}$ as

$$\hat{\boldsymbol{y}} = f_{\text{dec}}(\boldsymbol{c}). \tag{1.8}$$

The composite function $f_{\text{dec}} \circ f_{\text{enc}}$ is jointly trained end-to-end. One of the first and most prominent encoder-decoder models is the sequence-to-sequence model [139] where both the input and output data are sequences. Sequence-to-sequence models have become the state-of-the-art in neural machine translation, where the source sentence is processed by an encoder RNN, and the target sentence is produced using another decoder RNN. Another example of an encoder-decoder model is the autoencoder where an encoder function derives a latent representation of the input, and the decoder function reconstructs the input from the latent representation. Further examples of encoder-decoder architectures can, for example, be found in image caption generation [164], and clinical decision support systems [47, 48].

Model Training Neural networks are trained by minimizing the negative log-likelihood of the model parameters given the data $\mathcal{D} = \{\boldsymbol{y}^{(i)}, \boldsymbol{x}^{(i)}\}_{i=1}^{N}$, which contains pairs of input vectors $\boldsymbol{x}^{(i)}$ and labeled outputs $\boldsymbol{y}^{(i)}$. For i.i.d. training data the negative log-likelihood is

$$C = -\sum_{i=1}^{N} \log p(\boldsymbol{y}^{(i)} | \boldsymbol{x}^{(i)}, \boldsymbol{\Theta}), \tag{1.9}$$

with $\boldsymbol{\Theta}$ being a vector which contains all learnable parameters of the network. The concrete cost function C depends on the distribution of $p(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)}, \boldsymbol{\Theta})$. The most common cost functions are the squared error cost function, which is derived by assuming a Gaussian distribution, the binary cross-entropy cost function, which is derived by assuming a Bernoulli distribution, and the categorical cross-entropy cost function which is derived by assuming a Categorical distribution.

The cost function C is minimized using the back-propagation algorithm [124]. The back-propagation algorithm consists of three steps: The forward-pass, the backward-pass, and the parameter update. These three steps are iteratively repeated, until the cost converges to a minimum.

In the forward-pass, the network is evaluated given the input, while intermediate results are stored. We denote $\boldsymbol{z}^{(l)}$ as the output of a layer, before the activation function is applied, and $\boldsymbol{o}$ the output of the output-layer, before the output activation is applied. In the multilayer perceptron (as defined in Equation 1.1 and 1.2) these outputs are defined as

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}, \tag{1.10}$$

$$\boldsymbol{o} = \boldsymbol{A}\boldsymbol{h}^{(L)} + \boldsymbol{c}. \tag{1.11}$$

These intermediate results, as well as the activated outputs $\boldsymbol{h}^{(l)} = \psi(\boldsymbol{z}^{(l)})$ and $\hat{\boldsymbol{y}} = \phi(\boldsymbol{o})$ are stored, during the forward pass of the backpropagation algorithm.

In the backward-pass the so called error signals are computed for each layer. The error signals for each layer are defined as

$$\boldsymbol{\delta}^{(l)} = \nabla_{\boldsymbol{z}^{(l)}} C. \tag{1.12}$$

For the output layer, which we denote as layer $(L+1)$, the error signal is computed given the cost C and the output $\hat{\boldsymbol{y}}$ as

$$\boldsymbol{\delta}^{(L+1)} = \nabla_{\hat{\boldsymbol{y}}} C \odot \phi'(\boldsymbol{o}), \tag{1.13}$$

with $\odot$ denoting the element-wise vector product. The error signals of the subse-

quent hidden layers are computed recursively. For the MLP the error signals at the hidden layers are computed as

$$\boldsymbol{\delta}^{(l)} = ((\boldsymbol{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)} \odot \psi'(\boldsymbol{z}^{(l)}). \tag{1.14}$$

Given the activations $\boldsymbol{h}^{(l)}$ from the forward pass, and the error signals $\boldsymbol{\delta}^{(l)}$ from the backward pass, the partial derivatives for the weights can be computed efficiently. For the MLP the derivatives for the weights are

$$\frac{\partial C}{\partial \boldsymbol{W}^{(l)}(i,j)} = \boldsymbol{h}^{(l-1)}(j)\boldsymbol{\delta}^{(l)}(i), \tag{1.15}$$

and the derivatives for the bias terms are

$$\frac{\partial C}{\partial \boldsymbol{b}^{(l)}(i)} = \boldsymbol{\delta}^{(l)}(i). \tag{1.16}$$

All learnable parameters of the model are concatenated into the vector $\boldsymbol{\Theta}$ and the partial derivatives for all parameters are concatenated into the gradient vector $\nabla_{\boldsymbol{\Theta}} C$.

In the last step of every iteration of the backpropagation algorithm, the weights are updated according to the gradient descent update rule

$$\boldsymbol{\Theta}_{\text{new}} = \boldsymbol{\Theta} - \eta \nabla_{\boldsymbol{\Theta}} C. \tag{1.17}$$

The procedure is repeated iterativley until the cost function converges to a local minimum. The parameter η is called learning rate, and is a hyperparameter of the algorithm. Alternative update rules, such as Adam [81] or Adagrad [43], which adjust the learning rate dynamically for each update, have been proposed. In order to avoid the convergence of the cost function into a local minimum, or in situations where the dataset is too large, the parameter update is not performed on the full dataset $\mathcal{D}$. Instead, a random subset of the data, also called batch, is sampled in each iteration of the backpropagation algorithm. In this way, the calculated gradient is only an approximation of the complete cost function. By introducing this randomness, the optimization algorithm has a chance to jump out of a local minimum, and find a better solution. To avoid overfitting, the algorithm is often

not trained until convergence, but stopped, once the performance on a seperatly evaluated validation set, which has been hold out from the training data, does not increase. This technique is called early stopping.

1.2.3 Tensor Decompositions

Tensors can be considered as multi-dimensional arrays. Tensor decompositions are generalizations of matrix factorizations to higher order tensors. Low rank approximations provide latent representations for the involved statistical entities. Thus, tensor decompositions are considered a technique for learning representations. The first applications of tensor factorizations originate from psychometrics and chemometrics. Nowadays, tensor decompositions have also become increasingly popular in signal processing and machine learning. Surveys on the topic can be found in [82, 131]. In the following, we give a brief overview over the most important tensor decompositions.

CANDECOMP/PARAFAC The outer product of two vectors builds a rank-one matrix. Matrices with higher rank can be reconstructed from the sum of multiple rank-one matrices. The number of rank-one matrices needed for the reconstruction, determines the rank of the matrix. In a similar way, the outer-product of n vectors builds a n-mode rank-one tensor. The number of rank-one tensors, which are needed to sum to a n-mode tensor, defines its tensor-rank. Thus, a tensor can be decomposed into the sum of rank-one tensors, each being represented by the outer-product of n vectors.

For a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_S}$ the decomposition is

$$\boldsymbol{\mathcal{X}} \approx \sum_{r=1}^{R} \boldsymbol{a}_r^{(1)} \otimes \boldsymbol{a}_r^{(2)} \ldots \otimes \boldsymbol{a}_r^{(S)}, \tag{1.18}$$

with $\boldsymbol{a}_r^{(d)} \in \mathbb{R}^{n_d}$ and $\otimes$ denoting the outer-product. This decomposition has originally been proposed by Hitchcock [71]. Later it has been independently rediscovered by Carroll and Chang [67] as CANDECOMP decomposition and by Harshman [29] as PARAFAC decomposition. Thus, it is often referred to as CP (CANDECOMP/PARAFAC) decomposition. An additional scaling constant g_r for each of

the R components can be added. In an element-wise form the CP decomposition can be written as

$$\boldsymbol{\mathcal{X}}(i_1, i_2, ..., i_S) \approx \sum_{r=1}^{R} \boldsymbol{g}(r) \cdot \boldsymbol{A}_1(i_1, r) \cdot \boldsymbol{A}_2(i_2, r) \cdot \ldots \cdot \boldsymbol{A}_S(i_S, r), \tag{1.19}$$

where $\boldsymbol{A}_d \in \mathbb{R}^{n_d \times R}$ for $d \in \{1, ..., S\}$ are the so called factor matrices. The columns of the factor matrices correspond to the rank one components in Equation 1.18 and $\boldsymbol{g} \in \mathbb{R}^R$ contains the additional scaling constants. The decomposition is usually trained using the alternating least squares (ALS) algorithm, which iteratively updates one factor matrix per iteration. In each step, it solves the least squares problem which arises, when all factor matrices except one are being fixed.

The rows of the matrices $\boldsymbol{A}_1$ to $\boldsymbol{A}_S$ contain the vector representations for all entities along the axis of $\boldsymbol{\mathcal{X}}$. For example if $\boldsymbol{\mathcal{X}}$ is a three dimensional tensor representing *users* $\times$ *items* $\times$ *ratings*, the rows of $\boldsymbol{A}_1$ contain the latent representations for all users, the rows of $\boldsymbol{A}_2$ for each item, and the rows of $\boldsymbol{A}_3$ for each rating type. The length of the representation vectors is R, which is equal in all modes.

Tucker Decomposition Tucker [149] introduced a decomposition, which factorizes a tensor into a core tensor, and factor matrices for each mode. The core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{R_1, ..., R_S}$ weights all the interactions between the components of the factor matrices $\boldsymbol{A}_d \in \mathbb{R}^{n_d \times R}$ for $d \in \{1, ..., S\}$. The Tucker decomposition is defined as

$$\boldsymbol{\mathcal{X}}(i_1, i_2, ..., i_S) \approx \sum_{r_1=1}^{R_1} ... \sum_{r_S=1}^{R_S} \boldsymbol{\mathcal{G}}(r_1, ..., r_S) \cdot \boldsymbol{A}_1(i_1, r_1) \cdot \boldsymbol{A}_2(i_2, r_2) \cdot \ldots \cdot \boldsymbol{A}_S(i_S, r_S). \tag{1.20}$$

The factor matrices are sometimes constrained to be orthogonal. In this case the decomposition can be considered as a generalization to PCA. Thus, the decomposition is sometimes also referred to as n-mode PCA [85]. The Tucker decomposition can be computed using the truncated higher order singular value decomposition (HOSVD) algorithm, which derives the factor matrices by applying SVD to all mode-n matrix reshapes of the tensor. Given the factor matrices the core tensor can be computed using a closed form solution. Alternatively, the Tucker decom-

position can also be computed using an ALS algorithm.

The rows of the matrices $\boldsymbol{A}_1$ to $\boldsymbol{A}_S$ contain representations for the indexed entities similar to the CP decomposition. However, the fusion of the multiple representations is more powerful, as each possible combination of factors is weighted differently by an element of the core tensor $\boldsymbol{\mathcal{G}}$. Note that CP is a special case of the Tucker decomposition where the core tensor is diagonal. For higher order tensors the core tensor in the Tucker decomposition quickly explodes, as the number of elements in the core tensor grows exponentially with the number of dimensions.

Tensor Train Decomposition Due to the limitations on scalability of the Tucker decomposition, the tensor train decomposition for higher order tensors has been proposed. Tensor trains consist of multiple low rank tensors in a row, each with a fixed order of three so that the number of elements does not grow exponentially with the order of the input tensor. The tensor train decomposition is defined as

$$\boldsymbol{\mathcal{X}}(i_1, i_2, ..., i_S) \approx \sum_{r_0=1}^{R_0} ... \sum_{r_S=1}^{R_S} \boldsymbol{\mathcal{A}}_1(r_0, i_1, r_1) \cdot \boldsymbol{\mathcal{A}}_2(r_1, i_2, r_2) \cdot ... \cdot \boldsymbol{\mathcal{A}}_S(r_{S-1}, i_S, r_S), \tag{1.21}$$

where $\boldsymbol{\mathcal{A}}_d \in \mathbb{R}^{R_{d-1}, n_d, R_d}$ for $d \in \{1, ..., S\}$ and $R_0 = R_S = 1$ so that $\boldsymbol{\mathcal{A}}_1$ and $\boldsymbol{\mathcal{A}}_S$ are in fact matrices as the third dimension has only length one, and $\boldsymbol{\mathcal{A}}_2, \ldots, \boldsymbol{\mathcal{A}}_{S-1}$ are tensors of order three. Oseledets proposed a fast learning algorithm based on multiple successive singular value decompositions [110].

Note that the latent representations for the entities along the axes of $\boldsymbol{\mathcal{X}}$ are not vectors but matrices $\boldsymbol{\mathcal{A}}_d(:, i_d, :)$. Only entities of the first and last dimensions are represented by vectors. The latent matrix representations control the interaction to their neighboring entities.

General Multi-way Models Tensor decompositions learn representations for the involved entities in all dimensions of the input tensor. To approximate the original tensor the representations are fused according to the respective decomposition. A more general view on tensor decompositions can be formulated as

$$\boldsymbol{\mathcal{X}}(i_1, i_2, ..., i_S) \approx f(\boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_S}), \tag{1.22}$$

where the vectors $\boldsymbol{a}$ are the latent representations of the involved entities and f is the multi-linear tensor factorization, e.g., CP or Tucker. This more general formulation gives rise to the extension of the tensor decomposition framework to non-linear implementations of f, such as a multilayer perceptron. Using a neural network for f allows the modeling of arbitrary non-linear interactions between the learned latent factors. When a multilayer perceptron is used, the model is referred to as a multi-way neural network. Such a model has, for example, been used for knowledge graph modeling [41, 106], for clinical decision support systems [47] and for sensory data [50]. A non-linear approach using Gaussian Processes can be found in the Infinite Tucker decomposition [165].

1.2.4 Applications of Representation Learning

Representation learning has become a standard technique in many areas of machine learning. For example, matrix factorization has emerged to be the state-of-the-art method for collaborative filtering. In these settings, latent representations of users and items are derived by factorizing the adjacency matrix which describes previous interactions. Based on the similarity of the derived representations, new items are recommended to users. In particular, the Netflix challenge [83] has proven the superior performance of factorization techniques for collaborative filtering. For missing link prediction in large graph-structured knowledge bases, such as Yago [137], Freebase [22], or DBpedia [3], tensor factorization techniques have become a standard approach [106]. In these models, a latent representation is learned for every graph node and for the possible relations between the nodes. Based on these latent representations, a likelihood score for every possible triple can be derived. In this way, missing triples in the knowledge base are inferred. In general, tensor decompositions are a powerful method to learn representations of entities which are involved in relationships of higher orders. An overview of applications can be found in [82] and [131]. Latent representations obtained by tensor decompositions are also used for factor analysis in data mining [1, 67]. In natural language processing, learned word and sentence embeddings have become ubiquitous. They are

either learned unsupervised, based on co-occurrence of words in large text corpora [96], or simultaneously with a supervised model, e.g., for machine translation [140]. Representation learning is also used extensively in computer vision applications. Many of todays computer vision applications make use of CNNs, which hierarchically extract latent features from images. For example Facebook's DeepFace [142] searches images of the same person, using latent representations, which are derived from CNNs.

1.3 Introduction to Information Fusion

With the increasing ubiquity of large and diverse data collections, the question of how to improve knowledge discovery and decision making by combining information from multiple sources has gained increasing attention. According to Boström et al. the research field of information fusion can be defined as follows [24]:

> *"Information fusion is the study of efficient methods for automatically or semi-automatically transforming information from different sources and different points in time into a representation that provides effective support for human or automated decision making."*

The information sources can be as manifold as sensors, databases, simulations, ontologies, text documents, the Web, and humans. Thus, information fusion also comprises more traditional areas of research such as sensor fusion. By including not only different sources but also various points in time, the definition also includes estimation methods such as the Kalman filter.

Research on information fusion is concerned with a number of questions, such as [2]: Which sources and features to fuse? How to transform the data, and at which level of transformation to fuse the information? And which methods to use to efficiently achieve effective results? Thereby, information fusion research is strongly related to many other research areas such as machine learning, signal processing, robotics, multimedia analysis, and cognitive neuroscience. In this thesis, we focus on information fusion problems and techniques in the realm of machine learning research.

Information fusion is also central to cognitive neuroscience, as it plays a critical role in the human brain, e.g., [46, 55, 42]. Some evidence supports the notation that the brain processes and merges information in a hierarchical manner, e.g. when propagating information from the primary to the secondary sensory cortex. The superior colliculus is believed to fuse multi-modal inputs such as visual, haptic, or audio signals, to derive a percept. Furthermore, information in the brain is filtered and processed by mechanisms such as attention. Also, the integration of sensory inputs with prior knowledge about a situation seems very likely in the human brain [42].

In Section 1.3.1, we summarize the connections between information fusion and machine learning. Approaches to information fusion in machine learning can be broadly categorized into model-agnostic and model-based approaches, which we discuss in Section 1.3.2 and 1.3.3, respectively.

1.3.1 Information Fusion in Machine Learning

According to the definition provided by [24], the transformation of information from various sources and the effective support for decision making are central points in information fusion. Therefore, machine learning models represent an important set of methods for information fusion, as they are concerned with the transformation of information, e.g., feature extraction, representation learning and decision making, e.g. with a classification model. However, it is still subject to ongoing research on how to fuse data from multiple sources and explore their interactions. The machine learning models can include a wide range of tasks, such as clustering, novelty detection, or predictive modeling. The final decision, which is desired in the information fusion process, can be either directly obtained from the machine learning model or the machine learning model at least provides the necessary information for a separate decision step. The final decision step can be performed by a human or another automated process.

Machine learning aims at building statistical models to derive a semantic-level decision, given some inputs. In complex problem domains, the data might come from various sources. In this setting, the goal of information fusion is to gain better prediction accuracy by including multiple information sources.

One goal of information fusion in machine learning is more robust models, e.g., by combining evidence from multiple sources, or if for some channels only limited training data is available. Furthermore, the model can improve its performance if complementary information is captured in the different channels and cross-correlations can be exploited. Finally, in some cases, the problem of noisy or missing data channels can be overcome by the inclusion of information from multiple channels; see [15].

In what follows we give an overview of various scenarios that demonstrate information fusion problems in machine learning. However, it will also become clear that drawing a clear line between the various scenarios is not always possible.

Multi-view In multi-view learning, a particular data instance is represented by multiple views. These views can be derived from various sources or from a subset of features. For example, for the problem identifying a person, various views such as an image of the face, an iris-scan and a fingerprint can be used. In this case, the views are derived from various sources. Conversely, an image can be represented by a color view or a texture view. In this case, the various views are obtained by taking subsets of features. A survey of classical approaches to multi-view learning, including co-training, multiple-kernel learning, and subspace learning, can be found in [138]. More recent research concerns multi-view representation learning using autoencoders and deep learning techniques. A recent survey can be found in [158].

Multi-modal In multi-modal machine learning, the data involves multiple modalities, such as audio, video, or text. The various modalities are described by very different representations. For example, an image is represented by pixel values whereas text is represented by a sequence of words or tokens. Multi-modal machine learning includes the fusion of multiple modalities as input to a predictive model, e.g. audio-visual speech recognition, and the translation or alignment between various modalities, e.g., image caption generation. Many applications of multi-modal learning arise in the multimedia domain. A recent survey on the topic can be found in [15].

Multi-channel In multi-channel settings, the data is derived from various channels with the same modality. A popular example is electroencephalography (EEG) data, where the brain's electrical activity is measured at various positions; see [98, 130]. Other examples in which sensors are spatially distributed on a larger scale can be found in remote sensing [173] or sensor arrays such as multiple microphones in a room [148].

Multi-way Multi-way data is represented in a multidimensional array, e.g., EEG data, where the electrical activity is represented in three modes: channel, frequency, and time [1]. Each mode is split into a discrete set of options, which are described by the indices in the multidimensional array. In some applications, the data cannot be represented in a single multidimensional array but in several, where some of the modes are shared, e.g., an adjacency tensor of a knowledge graph of the form entity $\times$ relation $\times$ entity and a matrix containing side information about the entities of the form entity $\times$ features. Multi-way data appears in a wide range of application domains, such as psychometrics, chemometrics, text, and image analysis. A comprehensive review summarizing methods and applications of multi-way analysis can be found in [82]. More recent surveys with a focus on data fusion and machine learning applications are available in [113, 131].

The methods of information fusion in machine learning can broadly be categorized into model-agnostic and model-based information fusion approaches [15]. Model-agnostic approaches are independent of the specific machine learning method where-as in model-based approaches, the machine learning models are designed with regard to the information fusion problem at hand. We review the two approaches in the following sections.

1.3.2 Model Agnostic Information Fusion

Traditionally, information fusion in machine learning has been approached in a model agnostic way. In model agnostic fusion, the channels are fused independently of the machine learning model. The main advantage of model-agnostic fusion is that any generic machine learning model, such as linear- and logistic-regression,

decision trees, or support vector machines, can be applied without modifications to the model architecture. In the literature, model agnostic fusion is often divided into three strategies, which differ in the level where the information is merged.

Early Fusion In early fusion, also referred to as feature-level fusion, the information sources are fused at the raw feature level, often simply by concatenating the input features from multiple sources. The combined input features are then passed to a machine learning model, which can exploit the interactions between the features and outputs a prediction. For this approach, only a single model is necessary, which eases the training. However, the input size to the model can grow quite large. Thus, the early fusion approach is not scalable to a large number of input channels. Further problems can arise if the representations of the input channels are very heterogeneous, e.g., audio and image data. A single type of model, or in kernel systems, a single type of kernel, is often not suitable for all modalities, e.g., RNNs are better suited for modeling sequential data, whereas CNNs are more adequate for image data. In time-dependent data, the early fusion approach can also be problematic if the data from the input channels are not synchronized in time, as sequential models such as recurrent neural networks or hidden markov models assume fixed time scales. Some successful applications of early fusion can, for example, be found in [100, 127, 44].

Late Fusion In late fusion, also referred to as decision-level fusion, the information from various channels or modalities is combined at the decision level, i.e., the input features of each channel are processed by separate machine learning models. Each model makes a prediction, and the predictions are finally fused to a final decision. The fusion can be performed by rule-based methods such as voting, MIN, MAX, AND, OR, by a weighted combination, based on the uncertainty of the single models, by Bayesian inference, by the Dampster-Shafer evidence theory, or by training an additional fusion model, which takes the predictions of the single models as input and outputs the final prediction. Training multiple models allows for choosing the best model for each input channel, e.g., RNNs for text and CNNs for images. Furthermore, the late fusion approach can resolve problems with unsynchronized input channels, e.g., time series measured at various scales.

It is also easier to handle cases where the information from an input channel is missing. However, the main disadvantage of late fusion approaches is, that low-level interactions between the input features cannot be exploited, as each channel is processed independently. Also, including many models increases the training effort. Late fusion approaches have been widely applied; some successful examples can be found in [76, 51, 60, 97, 117, 115, 169].

Hybrid Fusion To benefit from the advantages of both techniques, hybrid approaches have been proposed. A hybrid approach can take a flexible combination of early and late fusion procedures for various input channels. Redundant architectures, where the various channels are input to a joint model (early fusion) and a separate model for each channel (late fusion), have proven advantageous. Successful applications using hybrid fusion include multi-sensor image analysis [105, 17], event detection in audio-visual data with external information [162, 86], and multi-modal speaker identification [161].

Both approaches have their advantages and disadvantages, which cannot be overcome in a model-agnostic fusion procedure. One way to overcome the problems is to use an early fusion approach but adjust the model so it overcomes some of the problems that arise when a generic model is used. We present these model-based methods in the next section.

1.3.3 Model-Based Information Fusion

In model-based fusion, the machine learning model is designed so that it supports the information fusion process by explicitly modeling the interactions between the channels. This arrangement allows for the inclusion of prior information about the system into the model architecture. In some cases, it also leads to more interpretable results by visualizing the information flow in the architecture and showcasing the importance of each channel for the prediction. In this section, we give an overview over the methods used in model-based fusion, with a focus on neural networks and tensor decompositions.

Neural Networks More recently, neural networks have been used for multi-channel and multi-modal modeling, as their architectures can easily be designed for problem-specific settings. When applying neural networks in a model-based fusion setting, the network architecture is designed so that it fits the specific fusion problem. It therefore reflects the a priori knowledge of how the processing pipeline should look. A popular framework is an extension to the encoder-decoder model, where multiple encoders learn a latent representation for the separate input channels and a decoder fuses the latent representations from the encoders. The idea behind this architecture is that the encoders extract meaningful higher-level latent features for each input channel, and the decoder learns the interaction between the channels based on the detected features. The encoders can be of various types, such as RNN and CNN. For example, if a vision and text modality are merged into a classifier, one could apply a CNN to the visual input and an RNN to the text, as these models derived expressive representations for the respective data types. For a classification, the decoder could then, for example, be of a feed-forward model type, which takes the combined representations from the encoders as an input. Thus, the information fusion takes place on a latent representation level. The whole network, including all decoders and the encoder, is typically trained end-to-end. In this way, the decoders should learn to directly extract the meaningful features for the respective task. However, one cannot be sure, that after the training, the separate processing steps work in the expected way. The intermediate features, which are extracted from the input channels, are typically not interpretable by humans. Also, the importance of the individual channels for the final prediction cannot be assessed. Modern neural network architectures often contain a large amount of parameters, allowing them to learn complex feature extractors and complex decision boundaries. However, they are also very expensive to train and require large amounts of training data.

One of the early successful examples of model-based information fusion using deep learning can be found in [103]. The authors used a restricted Boltzmann machine, which is a generative neural network model, for learning joint representations in audio-visual speech recognition by learning separate latent representations for each modality, which are combined into a joint latent representation. In [49], static and dynamic information about a patient in a clinical setting is fused

within a neural network architecture. The temporal information derived from the patient's former visits to the clinic is processed using an RNN whereas the static information is separately processed using a feedforward neural network. The latent representations derived from both parts are concatenated and fed into the decoder feedforward neural network, which predicts the best medication for the particular patient. The fusion of text and image data into a joint model can be found in recent visual question-answering models. Although various additional mechanisms are applied, most of the work follows the basic structure of learning a latent representation for the question using an RNN and a latent representation for the image using a CNN; see [159, 54, 163]. Further examples of model-based information fusion using deep learning techniques can be found in 3-D object recognition [45] and video-related tasks such as emotion recognition [79] and gesture recognition [102].

Tensor Decompositions Tensor decompositions are used in machine learning, data mining, and signal processing as a method for modeling multi-way data. The structure of the decomposition can be defined with respect to the input data and the specific problem at hand. Machine learning tasks built based on tensor decompositions include recommender systems, missing link prediction in knowledge graphs, and efficient regression, classification, and clustering of multi-way data [106, 113, 131]. When the data is represented in multiple tensors or matrices with common modes, they can be factorized jointly in a coupled tensor factorization. In coupled tensor factorization approaches, each tensor is factorized, and the common modes share the same representations. Coupled tensor decompositions have also been applied to multi-view learning, where each view is represented in a separate multidimensional array, and all arrays share the first mode, which describes the data instance [80, 88]. Successful applications include spatial temporal tensor regression [6], seizure prediction in EEG data [1], blind source separation in speech recordings [108], automatic conversation detection in emails [4], as well as image analysis, compression, and recognition [153, 154, 129].

Other Methods Many other methods are used for model-based information fusion. For example, probabilistic graphical models are well suited for model-based

information fusion. In probabilistic graphical models, the joint probability distribution of all involved variables, including a priori independence assumptions, is modeled, and the parameters are estimated from data. The structure of the model has to be determined beforehand and can thus be designed such that it reflects the prior assumptions about the modeled heterogeneous system. Special models have been proposed for modeling multi-modal data, e.g., [135, 65, 101]. Also, kernel systems such as support vector machines have been extended to model-based fusion methods. In multiple kernel learning, the features of each input channel are modeled with separate kernels, which are best suited for the respective modality; see [61]. Popular methods for parameter estimation in sensor fusion are the Kalman filter and non-linear methods such as the extended Kalman filter, unscented Kalman filter, and the particle filter. These methods are used for estimating the state of a system, e.g., the position of a moving object, by integrating uncertain measurements from either a single or multiple sensors; see [77].

1.4 Contributions of this Work

In this thesis, we propose various statistical models for advancing information fusion problems in machine learning. The contributions of this work lie in the intersection of representation learning and information fusion. Below, we briefly summarizes the main contributions of the thesis.

In Chapter 2, we propose a neural network architecture for fusing latent representations derived from multiple input channels. We further consider the problem of automatically determining what information to fuse in a dynamic multi-channel environment. Therefore, we apply the neural attention mechanism, which has become popular in language modeling, in a novel way, to address the dynamic information fusion problem. The effectiveness of the proposed architecture is evaluated on multiple real world sensor network datasets.

In Chapter 3, we propose a novel way for predictive modeling of multi-channel data using tensor decompositions. Specifically, we describe how the decomposition of high-dimensional and sparse data tensors can be used for classification and regression tasks. We then show a novel extension of the tensor decompositions to continuous inputs and model interactions between groups of input features. Our

experimental results on multiple datasets show that the efficient multi-linear models can reach similar performance as non-linear models, by reduced computational complexity.

In Chapter 4, we deal with the challenging problem of fusing semantic and sensory information. We propose novel machine learning models for this task based on the combination of tensor factorization for semantic modeling and deep learning models, which work well for modeling sensory data. Experiments on the task of visual relationship detection in images show promising results for this novel direction of research.

All significant contributions of this thesis have been published in conferences as peer-reviewed papers, as listed below.

[9] Stephan Baier, Denis Krompass, and Volker Tresp. Learning representations for discrete sensor networks using tensor decompositions. *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2016

[14] Stephan Baier and Volker Tresp. Factorizing sparse tensors for supervised machine learning. *NIPS workshop on tensor methods*, 2016

[12] Stephan Baier, Sigurd Spieckermann, and Volker Tresp. Attention-based information fusion using multi-encoder-decoder recurrent neural networks. *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2017

[13] Stephan Baier, Sigurd Spieckermann, and Volker Tresp. Tensor decompositions for modeling inverse dynamics. *Proceedings of the Congress of the International Federation of Automatic Control*, 2017

[10] Stephan Baier, Yunpu Ma, and Volker Tresp. Improving visual relationship detection using semantic modeling of scene descriptions. *International Semantic Web Conference*, 2017

[11] Stephan Baier, Yunpu Ma, and Volker Tresp. Improving information extraction from images with learned semantic models. *International Joint Conference on Artificial Intelligence*, 2018

I am the main author of all the listed publications. The papers have been written by me, and all the experiments have been conducted by me. At the beginning of each chapter we clearly state where the following contributions are published, and which parts are taken from the original publications.

Chapter 2

Attention-based Representation Fusion

In this chapter, we first propose a neural network architecture for fusing latent representations from multiple data channels. We then address the problem of how to dynamically determine what information to fuse. We therefore extend the proposed architecture using a self-attention mechanism, which automatically determines the importance of each data channel based on the current system state. We apply our model to the modeling of sensor networks, where we derive the latent representations for each sensor station using recurrent neural networks. The main contributions of this chapter are published in:

[12] Stephan Baier, Sigurd Spieckermann, and Volker Tresp. Attention-based information fusion using multi-encoder-decoder recurrent neural networks. *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2017

Sections 2.1, 2.2, 2.5, and 2.6 of this thesis correspond to Sections 1, 2, 3 and 4 of [12], but have been extended and edited to a large extent. Figure 2.1 has been copied and modified from [12]. Section 2.3 and 2.4 are entirely new. I am the main author of [12] . The paper has been written by me, and all the experiments have been conducted by me.

2.1 Introduction

Traditionally, multi-channel data is often processed using multivariate models, where the data from all channels is concatenated at the model input. From an information fusion perspective this corresponds to an early fusion approach. In this chapter, we propose a model-based fusion approach using neural networks. Our proposed architecture extends the popular encoder-decoder framework by applying dedicated encoders to each input channel. The latent representations from the different channels are then fused and fed into one or multiple decoder functions, which generate the predictions.

One problem in information fusion is that of determining which channels to fuse. An approach to this problem is to determine the cross-correlations between multiple channels and integrate those that show high cross-correlation. However, in dynamic systems, cross-correlations between different channels may vary in time. It is therefore desirable to also adjust the fusion process dynamically in time. Consequently we extend the proposed neural network architecture to incorporate an attention-based fusion layer that assesses the importance of the different input channels dynamically. Attention mechanisms have become popular in neural network research over recent years. We apply the attention in a novel way to address the dynamic fusion problem.

We apply our architecture to the modeling of distributed sensor networks, in which information from multiple data streams is combined. The sensor networks considered consist of multiple stations, where each station can measure multiple features at a single location. For each station, we implement an encoder function using recurrent neural networks. The latent representations from the dedicated RNN models are combined in the attenion-based fusion layer. After the representations are fused, they are passed to a decoder model which makes a prediction. We address the task of sequence-to-sequence prediction where the decoder network is another RNN that predicts the future behavior of a particular sensor station. Moreover, the proposed architecture can be easily generalized to other settings, such as classification or anomaly detection, by using different decoder functions.

Given the rising number of connected devices and sensors, often referred to as the Internet of Things (IoT), modeling sensor networks and multi-agent systems is

attracting increasing interest. We discuss the parallelizable nature of our proposed architecture in both training and inference contexts and show how this could be helpful when deploying the model in distributed environments.

We demonstrate the effectiveness of the multi-sequence-to-sequence network on three datasets. Two of these datasets are drawn from various sensor stations spread across Quebec and Alberta that measure climatological data. The third dataset contains energy load profiles from multiple zones of a smart energy grid. In our experiments on sensory data, we show that the proposed architecture outperforms both, purely local models for each agent as well as one central model of the whole system. This can be explained by the fact that the local sub-models learn to adapt to the peculiarities of the respective sensor station and, at the same time, integrate relevant information from other stations through the interconnection layer, which allows the model to exploit cross-correlations between the data streams of multiple stations.

The remainder of this chapter is organized as follows. In Section 2.2, we explain both the architecture of our proposed model and the attention-based fusion mechanism. Section 2.3 elaborates on the model's distributed training and inference. In Section 2.4, we discuss related work. Section 2.5 shows the experimental settings and results for the different experiments. Section 2.6 concludes our work and discusses possible directions of future research.

2.2 Representation Fusion Model

We propose a neural network architecture for modeling sensor networks consisting of multiple stations, where each station can potentially measure multiple features. The proposed architecture builds on top of the encoder-decoder framework and models each sensor station using a dedicated encoder function. We propose an attention-based interconnection layer that weights the latent representations from the encoders based on their importance for the prediction task. In this way cross-correlations between the single sensing stations can be exploited. A combined representation is then passed to a decoder model, which performs the prediction. It is also possible to add multiple decoder functions to the network. Due to the fact that all proposed layers are differentiable, we are able to train the whole system

of models end-to-end.

2.2.1 Multi-Encoder-Decoder RNNs

We consider the task of predicting multiple multivariate output sequences from multiple multivariate input sequences. The input sequences are represented by a three-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{E \times T_{\text{enc}} \times F_{\text{enc}}}$, where E denotes the number of encoder devices, T_{enc} denotes the encoder sequence length, and F_{enc} is the number of encoder features. Similarly, the output sequences are represented by a three-way tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{D \times T_{\text{dec}} \times F_{\text{dec}}}$, where D denotes the number of decoder devices, T_{dec} denotes the decoder sequence length, and F_{dec} is the number of decoder features. In the case of multivariate streaming data from a sensor network, the value $\boldsymbol{\mathcal{X}}(i,t,j)$ corresponds to the j-th feature measured at the i-th sensor station at time t. Similarly, the value $\widehat{\boldsymbol{\mathcal{Y}}}(i,t,j)$ corresponds to the predicted value of the j-th feature at the i-th output node at time t. If we consider, for example, the task of predicting the features of the next T_{dec} values for all stations in a sensor network, then D is the number of stations, F_{dec} is the number of features, and T_{dec} is the time period for which forecasts are performed. The input and output feature spaces may be identical, i.e. a prediction of all the sensor values per sensor node, or not, e.g., there may be a central control station making predictions for larger parts of the system.

We propose a neural network architecture that extends the encoder-decoder framework. The general architecture consists of multiple encoder functions, a interconnection layer, and possibly multiple decoder functions. Each input-sensing device is modeled by an encoder function

$$f_{\text{enc},i}(\boldsymbol{\mathcal{X}}(i,:,:)) = \boldsymbol{e}_i, \quad \text{with } i \in \{1,2,...,E\}, \tag{2.1}$$

which takes the data measured at the i-th sensing device as input and outputs a latent representation $\boldsymbol{e}_i$. For each output device, an interconnection function $f_{\text{con},j}$ combines the representations $\{\boldsymbol{e}_i\}_{i=1}^{E}$ as

$$f_{\text{con},j}(\{\boldsymbol{e}_i\}_{i=1}^{E}) = \boldsymbol{c}_j, \quad \text{with } j \in \{1,2,...,D\}. \tag{2.2}$$

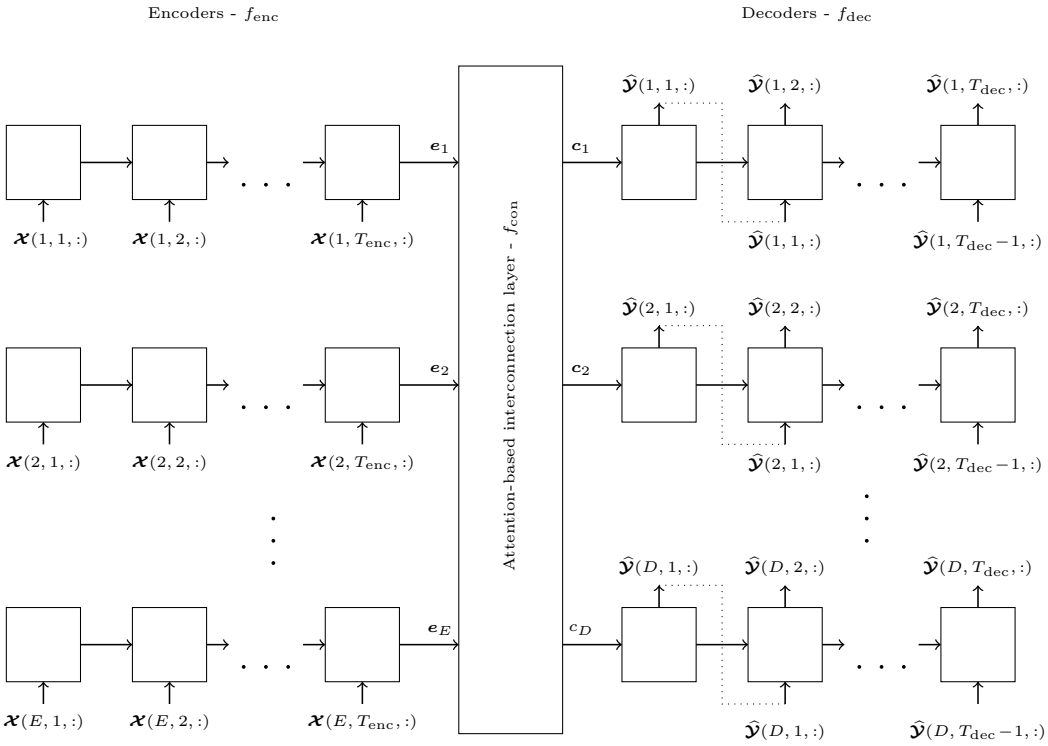


Figure 2.1: Unfolded multi-encoder-decoder recurrent neural network for multiple sequence-to-sequence prediction.

Finally, for each output device, a decoder function $f_{\text{dec},j}$ models the prediction using the respective combined representation $\boldsymbol{c}_j$ as

$$f_{\text{dec},j}(\boldsymbol{c}_j) = \widehat{\boldsymbol{\mathcal{Y}}}(j,:,:), \quad \text{with } j \in \{1,2,...,D\}. \tag{2.3}$$

In this way, information between the different input and output sequences can be exchanged through the interconnection layer.

In order to predict the sequence of future sensor behavior given the previous observations, we implement the functions f_{enc} and f_{dec} using recurrent neural networks. Thus, the functions are implemented as was explained in Equation 1.6 in Section 1.2.2. Figure 2.1 presents the architecture of a multi-encoder-decoder recurrent neural network model. For the sequence-to-sequence prediction, we model each encoder and each decoder function using an RNN. The rectangles describe the hidden states of the RNNs, while the arrows indicate the transformations be-

tween the states. Each encoder RNN iterates over the sequence produced by the respective sensing node. Thus, the input of the i-th encoder RNN is $\boldsymbol{\mathcal{X}}(i, t, :)$. We define the last hidden state of the i-th encoder RNN to be the encoder output $\boldsymbol{e}_i$. For each decoder RNN, a combined representation is computed by the respective interconnection function, which is represented by the large vertical box in the illustration. The interconnection layer outputs combined representations for each decoder. These representations are then used as initial hidden representation in the decoder RNNs. The decoder output $\widehat{\boldsymbol{\mathcal{Y}}}(i, t-1, :)$ is passed as an input to the next time step t of the i-th decoder RNN.

In order to maintain the scalability of the architecture the dimensionality of the merged decoder representation should be independent of the number of input channels. One canonical way to keep the dimensionality of the decoder representations the same size as the encoder representations is to use a sum or mean operation, such that

$$\boldsymbol{c}_j = \frac{1}{E} \sum_{i=1}^{E} \boldsymbol{e}_i \qquad \forall j \in \{1, \ldots, D\}. \tag{2.4}$$

This implementation for fusing the representation vectors, does not require additional learned parameters. Moreover, it can deal with a variable number of input channels, which is especially useful in settings where the number of input channels is not constant over time, e.g. moving devices where devices appear and disappear over time or where some input devices do not send any data, e.g., broken sensors. However, it does compute the same representation for all decoders. Thus, a more advanced implementation of the interconnection layer is desired, and will be described in the next section.

2.2.2 Spatial Attention Mechanism

We propose an implementation of the interconnection layer using an attention mechanism. In this way, the combination of latent representations is not fixed for every prediction but changes depending on the current context, which is encoded in the input representations. The attention mechanism assesses the importance of the representations of the encoding devices $\boldsymbol{e}_i$ and computes a weighted sum over

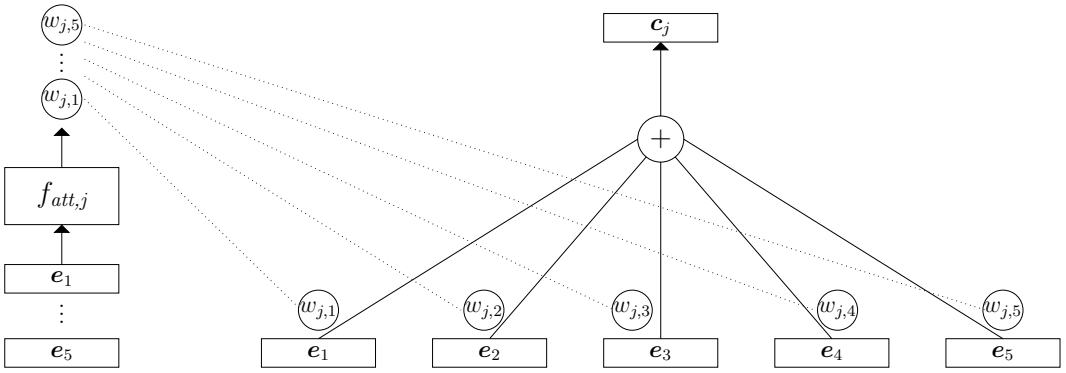


Figure 2.2: Illustration of the attention-based fusion layer with five encoder inputs and a single (the j-th) decoder output. The attention network on the left is shared for all inputs.

the latent vectors

$$\boldsymbol{c}_j = \frac{1}{E} \sum_{i=1}^{E} w_{ji} \boldsymbol{e}_i. \tag{2.5}$$

The vectors $\boldsymbol{e}_i$ are element-wise multiplied with the weight $w_{ij} \in \mathbb{R}$. To adjust the weights w_{ij} dynamically, we compute them using the additional attention function f_{att}. The attention function is implemented as a multi-layer percepron, as described in Equation 1.1 in Section 1.2.2. The outputs of the attention function are normalized through a softmax function, such that

$$z_{ji} = f_{\text{att},j}(\boldsymbol{e}_i), \tag{2.6a}$$

$$w_{ji} = \frac{\exp(z_{ji})}{\sum_{k=1}^{E} \exp(z_{jk})}. \tag{2.6b}$$

Whether or not attention is put on a representation $\boldsymbol{e}_i$ can vary for each prediction, depending on the information encoded in $\boldsymbol{e}_i$. The approach draws inspiration from the attention-based machine translation model [8]; however, the attention is not used across time, but spatially across sensing devices to address the dynamic fusion problem. The only parameters that need to be learned are those of the attention function. As all encoder representations are passed independently to the same attention function, the number of parameters is independent of the number of encoders, which yields a constant number of parameters.

Figure 2.2 illustrates the attention-based interconnection layer for an architecture with five encoders and a single decoder. The attention network on the left is specific to the decoder. All representations $\boldsymbol{e}_i$ are passed to the attention network separately. In practice, this can be implemented as a batch of inputs. For each encoder representation, a weight $w_1, \ldots, w_5$ is derived and then used in the fusion mechanism on the right. The output of the fusion mechanism is a weighted combination of the encoder representations, which is then passed to a decoder model.

Note that this mechanism can deal with a variable number of input devices. This is especially useful in settings where the number of input-devices is not constant over time, e.g., moving devices where devices appear and disappear over time, or where some input devices do not send any data, e.g., due to broken sensors.

2.2.3 Model Training

The model is trained end-to-end in a supervised fashion by minimizing the negative log-likelihood of a historical training set $\mathcal{D} = \{(\boldsymbol{\mathcal{X}}^{(n)}, \boldsymbol{\mathcal{Y}}^{(n)})\}_{n=1}^{N}$ w.r.t. the model parameters of all encoders and decoders. The log-likelihood of i.i.d. training examples can be written as

$$l = -\sum_{n=1}^{N} \log\ p(\boldsymbol{\mathcal{Y}}^{(n)} | \boldsymbol{\mathcal{X}}^{(n)}, \Theta), \tag{2.7}$$

where $\Theta = \{\Theta_{\text{enc},i}\}_{i=1}^{E} \cup \{\Theta_{\text{dec},j}\}_{j=1}^{D}$ is the set of parameters for all encoders and decoders. We assume the parameters of the j-th interconnection function to be part of $\Theta_{\text{dec},j}$. Thus, we obtain

$$l = -\sum_{n=1}^{N} \log\ p(\boldsymbol{\mathcal{Y}}^{(n)} | \boldsymbol{\mathcal{X}}^{(n)}; \{\Theta_{\text{enc},i}\}_{i=1}^{E}, \{\Theta_{\text{dec},j}\}_{j=1}^{D}). \tag{2.8}$$

Due to the conditional independence of the decoders, we can simplify (2.8) to

$$l = -\sum_{j=1}^{D} \sum_{n=1}^{N} \log\ p(\boldsymbol{\mathcal{Y}}^{(n)}(j,:,:) | \boldsymbol{\mathcal{X}}; \{\Theta_{\text{enc},i}\}_{i=1}^{E}, \Theta_{\text{dec},j}) = \sum_{j=1}^{D} l_j, \tag{2.9}$$

obtaining a sum of negative log-probabilities that we refer to as $l_1, ..., l_D$. Since all components in the complete encoder-decoder architecture are differentiable, the network can be trained via backpropagation. By training the complete model end-to-end, each encoder learns to encode the relevant information for all decoding tasks, rather than solely learning a good model for the specific input sequence.

2.3 Distributed Settings

Many modern field devices are equipped with powerful hardware for computation and the ability to exchange data with each other. To make use of these capacities, it is sometimes desirable to run analytics models directly in these decentralized environments. Our proposed architecture fits with this paradigm, as it is possible for the single decoder RNNs to run directly on the local sensing devices. When predictions are being made for each device, the decoder models can also run locally. In this section, we demonstrate what the parallel training and inference of the model could look like in a distributed environment.

2.3.1 Parallel Training

We train the parameters Θ using backpropagation and stochastic gradient descent. To update the parameters in each iteration, we need to compute the gradient of the cost function w.r.t. the network parameters.

Decoder Training The overall gradient w.r.t. the decoder parameters can be split into the sum of gradients of the single decoder likelihoods as

$$\nabla_{\Theta_{\text{dec}}} l = \sum_{j=1}^{D} \nabla_{\Theta_{\text{dec}}} l_j. \tag{2.10}$$

The gradient w.r.t. $\Theta_{\text{dec},j}$, $j \in \{1, 2, ..., D\}$, is zero in all l_j with $j \neq i$. Thus, the gradients w.r.t. all decoder weights can be computed independently and, hence, concurrently, for each decoder as

$$\nabla_{\Theta_{\text{dec},j}} l = \nabla_{\Theta_{\text{dec},j}} l_j. \tag{2.11}$$

for each $j \in \{1, ..., D\}$ **do** ▷ Parallel Decoders
  update($\Phi_{\text{dec},j}$)
  broadcast($\{\boldsymbol{\delta}_{i,j}\}_{i=1}^{E}$)

for each $i \in \{1, ..., E\}$ **do** ▷ Parallel Encoders
  $\{\boldsymbol{\delta}_{i,j}\}_{j=1}^{D} \leftarrow$ collectDeltas()
  update($\Phi_{\text{enc},i}$)

Algorithm 1: Multi-Enc-Dec RNN Backpropagation

Encoder Training Computing the gradients w.r.t. the encoder weights requires backpropagation of the error signal through the decoders' local copies of the encoder representations $\boldsymbol{e}_i$ that is performed concurrently on all decoder devices, such that

$$\boldsymbol{\delta}_{i,j} = \nabla_{\boldsymbol{e}_i} l_j. \tag{2.12}$$

Backpropagation continues on each encoder device i by collecting the local copies of the i-th encoder representation error signals $\boldsymbol{\delta}_{i,j}$ from all decoders j and summing them, such that

$$\boldsymbol{\gamma}_i = \sum_{j=1}^{D} \boldsymbol{\delta}_{i,j}. \tag{2.13}$$

The error signal $\boldsymbol{\gamma}_i$ relates to the encoder representation $\boldsymbol{e}_i$. Backpropagation continues using the chain rule in order to compute the gradients w.r.t. the encoder weights $\Theta_{\text{enc},i}$ where $\Theta_{\text{enc},i}(k)$ denotes the k-th variable of $\Theta_{\text{enc},i}$, such that

$$\frac{\partial l}{\partial \Theta_{\text{enc},i}(k)} = \sum_{r=1}^{\dim(\boldsymbol{e}_i)} \boldsymbol{\gamma}_i(r) \frac{\partial \boldsymbol{e}_i(r)}{\partial \Theta_{\text{enc},i}(k)}. \tag{2.14}$$

Since we can compute the gradient of the cost w.r.t. the encoder weights $\Theta_{\text{enc},i}(k)$ concurrently on each encoder device once the error signal $\boldsymbol{\gamma}_i$ is available, the weight updates are performed concurrently on all encoders. Algorithm 1 shows the training procedure for one update iteration in pseudo code.

for each $i \in \{1, ..., E\}$ **do** ▷ Parallel Encoders
  $\boldsymbol{e}_i \leftarrow f_{\text{enc},i}(\boldsymbol{\mathcal{X}}(i, :, :))$
  broadcast($\boldsymbol{e}_i$)

for each $j \in \{1, ..., D\}$ **do** ▷ Parallel Decoders
  $\{\boldsymbol{e}_i\}_{i=1}^{E} \leftarrow$ collectStates()
  $\boldsymbol{c}_j \leftarrow f_{\text{con},j}(\{\boldsymbol{e}_i\}_{i=1}^{E})$
  $\widehat{\boldsymbol{\mathcal{Y}}}(j, :, :) \leftarrow f_{\text{dec},j}(\boldsymbol{c}_j)$
  return $\widehat{\boldsymbol{\mathcal{Y}}}(j, :, :)$

Algorithm 2: Multi-Enc-Dec-RNN Inference

2.3.2 Parallel Inference

When running in a streaming environment, predictions can be performed at their own rate, which can be smaller than the sampling rate. Algorithm 2 shows how the model is executed in a distributed environment. Each sensing device computes a representation, which encodes the relevant information of preceding measurements in a single vector representation $\boldsymbol{e}_i$. As we have defined $\boldsymbol{e}_i$ to be the latest hidden state of the encoder RNN, f_{enc} can run in a streaming manner over the input data without having to recall previous measurements. This can be done concurrently for each sensing device. The resulting encoder representation $\boldsymbol{e}_i$ is broadcast to all decoder devices at the time a prediction is requested. The decoder devices simultaneously collect the encoder representations, combine them using their respective interconnection functions and output a prediction. Thus, the architecture scales along with the number of devices as the computational resources of the devices are exploited. When the sampling rate is higher than the prediction rate, or when the dimensionality of the representations is smaller than the number of measured features, the distributed model compresses information by sending latent representations instead of raw data. In this way, communication is more efficient in the decentralized environment compared to a centralized solution, where all measurements need to be sent to a central server.

2.4 Related Work

Our proposed model is an extension of the general encoder-decoder framework that has become popular for various machine learning tasks, including machine translation, image caption generation, and automatic speech recognition [140, 32]. The idea of using multiple encoders and decoders has also recently been considered in natural language processing [92, 52, 66]. We propose an interconnection layer that joins the latent representations of all encoders using an attention mechanism. Thereby, the attention mechanism, which was originally developed for neural machine translation [8, 32], is applied in a novel context. Typically, the attention mechanism is applied in sequence-to-sequence models to the individual steps of the decoder RNN; in our model, however, it is applied to dynamically fuse the representations from multiple encoders. Attention mechanisms have been further developed in many directions. For example, a neural network architecture for sequence-to-sequence prediction has recently become popular, which solely uses attention, without any recurrent structures [155]. Dropping the recurrent units, allows for massive parallelization of the model.

Some standard approaches to modeling multivariate time series data include multivariate linear regression and multivariate ARMA and ARIMA models. Box and Jenkins proposed a standard procedure for fitting these models [25]. Recurrent neural networks can be interpreted as nonlinear ARMA models [37]. One disadvantage of using neural networks is that they are completely deterministic. In many practical use cases, probabilistic models such as the Kalman filter and its non-linear extensions are preferred over deterministic approaches, because an estimate of the models uncertainty is desired. Traditionally, the selection of which time series to include in a multivariate model is accomplished by computing the cross-correlations on a historic dataset. Channels that show low cross-correlations with the target channel are not included in the predictive model. This approach, however, is only able to reliably detect linear relationships.

In signal processing, the sensor stream selection is typically formulated as an optimization problem. The goal is to find the best subset of sensor streams based on some relevant criterion. For large numbers of sensor streams, however, this problem becomes infeasible. Therefore, heuristics have been proposed for solving

this problem, e.g., [111, 78]. In [160], a procedure for finding the optimal combinations of modalities for fusing multimedia data is proposed. An overview of methods for selecting the right modalities in fusion problems can be found in [2].

2.5 Experiments

We evaluate the performance of the multi-encoder-decoder network using sequence-to-sequence prediction in sensor networks on two climatological datasets and a smart grid dataset. The chosen task is the prediction of future network behavior given a sequence of past measurements. Predictions are made for every sensor station and all features; thus, $E = D$ and $F_{\text{enc}} = F_{\text{dec}}$.

2.5.1 Datasets

For the first two datasets, we consider the modeling of spatially distributed weather stations. The multi-channel setting gives rise to our proposed model-based fusion architecture. The cross-correlation change over time based on the climatological conditions; this motivates the use of our proposed dynamic fusion architecture. We consider a sensor network of environmental sensing stations measuring climatological data on an hourly basis. The first dataset consists of 18 stations distributed across Quebec, each measuring air temperature, dew point, relative humidity and wind speed. The second dataset is a sensor network of 15 environmental sensors spread across Alberta and measuring the same features. We downloaded 5 years of data (between 2010 and 2014) from ASOS[1] and selected stations and features with the lowest number of missing values. We extracted sequences of 72 hours in length as input sequences and used the subsequent 24 hours as target sequences using a sliding window. After filtering missing values, we retained a total of 26991 sequence-to-sequence pairs in the Quebec data set and 34427 pairs in the Alberta data set. The data was split into training, validation and test sets. The data gathered between 2010 and 2013 was used for training (with 5 percent of it used for validation), while the data from 2014 was used for testing the models. This

[1] https://mesonet.agron.iastate.edu/request/download.phtml

resulted in a test data size of 26991 pairs in the Quebec data set and 3507 pairs in the Alberta dataset.

The third dataset considers short-term load forecasting in a power grid. Many new sensor measurements have recently become available due to the introduction of smart grid technology. Short-term load forecasting is an important problem in the utility industry with impacts on generation, transmission, distribution, and retail (see [74]). We use the load prediction data set from [74], which measures hourly loads in kW, and predict the load profiles for the next 3 days on the basis of the previous 21 days (i.e. 3 weeks). We selected 18 zones with historical load profiles. The data encompasses 4.5 years of data gathered between 2004 and 2008. We set the time rate of the prediction system at one day. For each day we consider 24 features, which are the loads at each hour. After removing sequence pairs with missing values, we obtain a dataset of 1187 training samples and 169 test samples. We use 5 percent of the training data for validation of different hyperparameter settings.

2.5.2 Experimental Setting

We evaluate various models based on their mean squared error on the test set. We normalized the data to have a mean of zero and a standard deviation of one. This results in a baseline mean squared error of 1.0 for a constant prediction of the mean. We further report the constant prediction of the last observed value as a baseline for each measured feature. We compare our model to linear auto-regression, which is a linear regression model that takes the previous measurements as input. The simple auto-regression models have shown superior performance in the task of energy load forecasting [74] compared to more complicated ARIMA models. We also conduct comparisons against regular RNN models. Both baseline models are trained in two different settings: (i) a separate model for each station, i.e., no cross-correlations can be exploited and (ii) a joint model for all stations, i.e., cross-correlations between stations can be exploited. For all RNN models, we performed the experiments with the basic model, a GRU-RNN, and an LSTM-RNN. The third dataset is too small to train the GRU and LSTM extensions without overfitting, as these models are very parameter-intensive. We

	Quebec			Alberta		
Model	Basic	GRU	LSTM	Basic	GRU	LSTM
Last observed values	0.6515	-	-	0.7295	-	-
Linear regression per station	0.4289	-	-	0.4189	-	-
Linear regression all stations	0.3562	-	-	0.3487	-	-
Enc-dec-RNN per station	0.3817	0.3728	0.3877	0.3492	0.3453	0.3581
Enc-dec-RNN all stations	0.3477	0.3488	0.3540	0.3468	0.3389	0.3505
Multi-enc-dec RNN	0.3361	0.3481	0.3426	0.3381	0.3387	0.3490
Multi-enc-dec RNN attention	0.3328	0.3314	0.3403	0.3289	0.3348	0.3485

Table 2.1: Mean squared error results for the climatological test sets.

compare these baseline models against our proposed architecture. The first model we evaluate is the multi-encoder-decoder with the simple mean interconnection layer (as shown in Equation 2.4). The second model is the extension proposed in Section 2.5 with the spatial attention mechanism. To optimize the neural network models, we used the stochastic gradient descent variant Adam [81], which uses an adaptive learning rate for each parameter. To avoid overfitting, we stopped training when the error on the validation set did not decrease within 10 iterations. A hyperparameter search was conducted on the validation set, including parameters such as learning rate, size of hidden units and regularization rate. All experiments were implemented using Theano [143] and Keras [35]. We run the models on a computer with a dual core Intel i5 CPU with 2.6 GHz and 16 GB of memory.

2.5.3 Experimental Results

Table 2.1 shows the results for both climatological datasets. The optimal size of the hidden state of the multi-encoder-decoder models was found to be 130 for the basic model, 70 for the GRU and 100 for the LSTM models. With larger hidden states the performance on the validation set decreased due to overfitting. For the RNN, which models all stations jointly, a larger hidden state of 300 for the basic model and 250 for the GRU and LSTM models was necessary to yield optimal results. Compared to dedicated models for each station, we can see that both the RNN and linear models perform significantly better when all stations are integrated into one model. This observation indicates strong cross-correlations between the stations.

Model	MSE
Last observed values	0.5169
Linear regression per station	0.3382
Linear regression all stations	0.3164
Enc-dec-RNN per station	0.3150
Enc-dec-RNN all stations	0.2956
Multi-enc-dec RNN	0.3020
Multi-enc-dec RNN attention	0.2884

Table 2.2: Mean squared error results for the smart grid data test set.

The use of individual RNNs per station yields better performance than the linear regression model per station, while the joint RNN for all stations outperforms the linear model for all stations. The multi-encoder-decoder RNN has the exact same number of parameters as the multivariate RNN for all stations. Experimental evaluation shows that using the fusion architecture improves the results slightly. When adding the proposed attention mechanism for the information fusion, the mean squared error decreases even further for both climatological datasets. We further observe that GRUs improve the prediction in some cases, whereas LSTMs do not perform better than the basic RNNs. In [57], it has also been found that LSTMs are not particularly well suited for time series forecasting.

The results for the smart grid data set are reported in Table 2.2. Here, the prediction of the load profile for the last day (last observed values) is already a good baseline, as the profiles do not change drastically within three days. The linear model with all stations included yields slightly improved predictions relative to the single models. Moreover, the RNN model including all stations outperforms the single per-station RNN models. The distributed multi-encoder-decoder model achieves similar or better results than the standard RNN model, depending on the implementation of the interconnection layer. The multi-encoder-decoder model with the mean interconnection layer and the RNN model including all stations jointly perform very similar, whereas the attention-based interconnection layer clearly outperforms these simpler models. We also tested GRUs and LSTMs on this dataset but discovered that the number of training examples was too small at 1187 to allow these parameter-intensive models to generalize well. Again, we

found a hidden state size of 130 units for the multi-encoder-decoder RNN yielded best results. For the RNN, that models all stations, 130 hidden units were also sufficient.

2.6 Conclusion

In this chapter, we proposed a model-based information fusion architecture, which extends the popular encoder-decoder framework. Our neural network architecture combines latent representations derived from multiple encoder functions and feeds them to the decoders, which then generate the predictions. We further extend the neural network architecture for dynamic information fusion. The problem of what channels to fuse is an important research question in information fusion. Our proposed approach carries out the selection based on data and dynamically adapts to changes in the system. The fusion of hidden representations of multiple encoder networks, using an attention mechanism, allows for the exploitation of cross-correlations across channels. For the dynamic adoption of the fusion process, we proposed the attention-based interconnection layer, which efficiently learns to combine information from multiple sensor stations by drawing attention to the stations containing information relevant to the current prediction. The attention varies based on the current context. To the best of our knowledge, this is a novel method in the context of intelligent sensor fusion. Using end-to-end training, the complete model (consisting of the encoders, the interconnection layer with an attention mechanism, and the decoders) is trained to predict a sequence of future behavior. It is important to note that the encoders are not trained to be a good model of the input sequence, but rather to produce a representation that leads to good output sequence predictions.

We showed how the model could be deployed in a distributed environment, such that local computational resources (which are available in most modern sensing devices) can be exploited. This has potential applications in real-world sensor networks such as smart grids or remote sensing. An important property of RNNs, which we use as encoder models, with regard to data stream modeling is their ability to selectively encode information from previous time steps in a compact hidden state representation that is updated continuously at every time step. Thus,

RNNs are good candidates for the efficient processing of streaming data.

In various experiments on sensor network data, the multi-encoder-decoder model showed clearly better results relative to a standard RNN, which models all stations jointly in a single model. This might be due to the dedicated encoder functions for each sensor station, which learn to encode the optimal information for the predictions, and the decoders ability to decide which representations are important for the current prediction.

The proposed architecture can easily be extended to different prediction tasks such as classification or anomaly detection. It is also possible to integrate different neural network architectures, such as convolutional neural networks or feed-forward neural networks, for both encoders and decoders.

Chapter 3

Discriminative Tensor Decompositions

In this chapter, we consider modeling the interactions between groups of input features using tensor decompositions. We first formulate an approach for applying tensor decompositions to supervised discriminative modeling with discrete input features. We then generalize the approach to continuous inputs and apply the model to the application of modeling inverse dynamics. The main contributions of this chapter are published in:

[9] Stephan Baier, Denis Krompass, and Volker Tresp. Learning representations for discrete sensor networks using tensor decompositions. *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2016

[13] Stephan Baier, Sigurd Spieckermann, and Volker Tresp. Tensor decompositions for modeling inverse dynamics. *Proceedings of the Congress of the International Federation of Automatic Control*, 2017

Section 3.1 is taken from Section 1 in [9] and changed broadly. Section 3.2 is partly taken from Section 4 in [9] and the Sections 2 and 3 in [13]. Subsection 3.2.3 is entirely new. The Sections 3.3, 3.4, and 3.5 are taken from the Sections 2, 5, and 6 in [9], and the Sections 4, 5, and 6 in [13], respectively. All sections haven been edited to a large extend. Figure 3.2 has been published in [9] and Figure 3.4 and

Figure 3.5 have been published in [13]. I am the main author of the papers [9] and [13]. The papers have been written by me, and all the experiments have been conducted by me.

3.1 Introduction

Within recent years, tensor decompositions have found a number of applications in machine learning, e.g., modeling knowledge graphs [106], weight compression in neural networks [109], and spatio-temporal regression [6]. In this chapter, we propose a method that exploits the inherent multi-way structure of tensor decompositions for discriminative modeling. Tensor decompositions model the interactions between various inputs by fusing learned latent representations of each input. In our approach the data is mapped to a high dimensional sparse tensor, i.e., most values are unobserved, and the tensor is then decomposed using common tensor factorization techniques. In particular, the CP and tensor train decompositions have shown to be applicable for the decomposition of tensors with many dimensions.

We first consider the problem of fusing information from various categorical variables. This setting can, for example, be found in the technical application of modeling sensor networks with multiple input sensors each measuring values on a discrete scale. By applying the tensor decompositions, a representation is learned for all possible measurements of all sensors. These representations are fused in order to classify the current behavioral state of the whole sensor network. The decomposed tensor represents the space of all possible combinations of sensor values. By learning a representation for each possible value of all sensors, the decomposition allows for approximating highly non-linear functions. The proposed models can be generalized to different distributions of the output data by applying different activation functions and different cost functions, similar to generalized linear models. We evaluate the performance of the proposed tensor models with discrete inputs on various datasets from the UCI data repository [40]. The experimental evaluation shows that the tensor decomposition models reach similar accuracy levels as support vector machines, which is a popular non-linear machine learning model for this kind of problem, while maintaining lower runtime complexities. We

further show that interpretability measures, such as odds ratios, can be computed efficiently for the tensor models.

We then propose a generalization to continuous inputs, by explicitly learning a mapping from the input data to the latent representations. In this way, a continuous version of tensor decompositions is derived. We apply this extension to learn an inverse dynamics model, that computes the necessary joint torques of a robot's motors for the execution of a desired movement. We group the desired joint positions, velocities, and accelerations of all degrees of freedom of the robot, resulting in a tensor of order three, which can be modeled using the Tucker decomposition. Our model exploits the inherent three-way interaction of *positions* $\times$ *velocities* $\times$ *accelerations*. We evaluate our model on a dataset of a seven degrees of freedom SARCOS robot arm that was introduced in [156]. An inverse dynamics model is learned based on collected trajectories, and its performance is evaluated on a test set. The results show that our model outperforms a number of competitive baseline methods, such as linear regression, radial basis function networks (RBF-networks), and support vector regression. Furthermore, the Tucker model shows superior performance over a CP model. Our proposed model gains similar results as the state-of-the-art methods on this task, but at simultaneously significantly shorter training and inference times. In this application, the inference time specifically matters, as the model needs to be deployed in a real-time control setting.

This chapter is organized as follows. In Section 3.2, we introduce our approach, applied in regard to discrete input data and the extension to continuous input data. We further present the generalizations to different output distributions, and discuss the efficient computation of interpretability measures, such as the odds ratio. In Section 3.3, we discuss related work, while Section 3.4 includes experiments on multiple discrete classification tasks and the application to modeling inverse dynamics. We conclude this chapter in Section 3.5.

3.2 Tensor Decompositions for Discriminative Modeling

In this section, we show how the decompositions of sparse and higher-order tensors can be applied to discriminative machine learning. We discuss their characteristics and application settings.

3.2.1 Predictive Model

Tensors and tensor decompositions can be considered as functions of multiple indices. In the full tensor, a combination of input indices maps directly to the corresponding function value. In the decomposed form, each input index maps to a latent representation. For deriving the function value, the latent representations are combined in a polynomial model, which is determined by the form of decomposition model. In this way, tensor decompositions are able to model the interactions between the input indices. In the following, we use the decomposition function for modeling the conditional probabilities in discriminative machine learning settings.

We consider a discriminative machine learning problem with $S \in \mathbb{N}$ discrete input variables. Each of the input variables x_j for $j \in \{1, \ldots, S\}$ assumes one out of $F_j \in \mathbb{N}$ discrete values. Furthermore, we consider a dependent variable y which we for now assume to be $y \in \mathbb{R}$. We model a function for a dataset of N training examples $\{y^{(i)}, (x_1^{(i)}, \ldots, x_S^{(i)})\}_{i=1}^N$. All training examples are mapped to a sparse tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{F_1,\ldots,F_S}$. The tensor is filled with

$$\boldsymbol{\mathcal{Y}}(x_1^{(i)}, \ldots, x_S^{(i)}) = y^{(i)} \quad \forall i \in \{1, ..., N\}. \tag{3.1}$$

The remaining entries of the tensor, which do not occur in the training data, are left unknown. This results in $\boldsymbol{\mathcal{Y}}$ being a very sparse tensor. We approximate the tensor $\boldsymbol{\mathcal{Y}}$ using a low-rank tensor decomposition. When using the CP decomposition we derive

$$\boldsymbol{\mathcal{Y}}(x_1, \ldots, x_S) \approx \sum_{r=1}^{R} \boldsymbol{g}(r) \cdot \boldsymbol{A}_1(x_1, r) \cdot \boldsymbol{A}_2(x_2, r) \cdot \ldots \cdot \boldsymbol{A}_S(x_S, r) = \hat{\boldsymbol{\mathcal{Y}}}(x_1, \ldots, x_S). \tag{3.2}$$

Imposing a low rank R, the approximation results in a dense tensor $\hat{\boldsymbol{\mathcal{Y}}}$. It describes the outcome $\hat{y}$ for all combinations of the input variables $(x_1, ..., x_S)$. However, it would be impossible to compute and store the whole approximated tensor $\hat{\boldsymbol{\mathcal{Y}}}$. Thus, only the parameters of the decomposition are stored. When predicting $\hat{y}$ for a new set of input variables, the representations for that tuple are indexed, and the approximation is computed on demand.

If we apply the Tucker decomposition, which considers all interactions between the latent representations, we get

$$\boldsymbol{\mathcal{Y}}(x_1, \ldots, x_S) \approx \sum_{r_1=1}^{R_1} \ldots \sum_{r_S=1}^{R_S} \boldsymbol{\mathcal{G}}(r_1, \ldots, r_S) \cdot \boldsymbol{A}_1(x_1, r_1) \cdot \boldsymbol{A}_2(x_2, r_2) \cdot \ldots \cdot \boldsymbol{A}_S(x_S, r_S). \tag{3.3}$$

The core Tensor $\boldsymbol{\mathcal{G}}$ grows exponentially with the number of dimensions S. For machine learning problems with many input variables, the core tensor becomes to large, thus the decomposition is not suited for these kind of problems. Another scalable decomposition is the tensor train decomposition, which is

$$\boldsymbol{\mathcal{Y}}(x_1, x_2, \ldots, x_S) \approx \sum_{r_0=1}^{R_0} \ldots \sum_{r_S=1}^{R_S} \boldsymbol{\mathcal{A}}_1(r_0, x_1, r_1) \cdot \boldsymbol{\mathcal{A}}_2(r_1, x_2, r_2) \cdot \ldots \cdot \boldsymbol{\mathcal{A}}_S(r_{S-1}, x_S, r_S), \tag{3.4}$$

with $\boldsymbol{\mathcal{A}}_d \in \mathbb{R}^{R_{d-1}, F_d, R_d}$ for $d \in \{1, \ldots, S\}$ and $R_0 = R_S = 1$ so that $\boldsymbol{\mathcal{A}}_1$ and $\boldsymbol{\mathcal{A}}_S$ become matrices. Thus the entities of the first and the last dimension are represented by vectors, where as the entities of all other dimensions are represented by matrices. To reduce the number of hyperparameters, the length of the representations R_1 to R_{S-1} in the tensor train decomposition are assumed equal in this work.

Model Analysis The tensor decompositions describe multi-linear functions, which are fast to compute. Table 3.1 summarizes the runtime and space complexities of the different tensor models. When assuming multiplication and summation operations being performed in constant time, the time complexity for evaluating the CP model is $\mathcal{O}(R \cdot S)$, as it needs $R - 1$ summations and $R \cdot S$ multiplications. Similarly, the time complexity for the tensor train model is $\mathcal{O}(R^2 \cdot S)$ as it

Method	Time Complexity	Space Complexity
CP	$\mathcal{O}(R \cdot S)$	$\mathcal{O}(R \cdot F \cdot S)$
tensor train	$\mathcal{O}(R^2 \cdot S)$	$\mathcal{O}(R^2 \cdot F \cdot S)$
Tucker	$\mathcal{O}(R^S \cdot S)$	$\mathcal{O}(R^S + R \cdot F \cdot S)$

Table 3.1: Comparison of the runtime complexities of different tensor decomposition models.

needs $R^2(S-2)+R$ multiplications and $(R-1)\cdot R\cdot(S-2)+(R-1)$ summations. The Tucker decomposition has a time complexity of $\mathcal{O}(R^S \cdot S)$, with $R^S \cdot S$ multiplications and R^S summations.

The relation of the proposed tensor decomposition models to linear regression with polynomial interactions is the following. The tensor decompositions can be interpreted as a factorization of the F^S coefficients of a polynomial regression model, which models all joint interactions of the input variables. Through the factorization the number of parameters is significantly reduced. The representation for a certain input is the same for all combinations where this input appears in. This coupling allows the model to generalize to combinations, which have not been observed in the training data. In this way, the parameters are reduced to $R \cdot F \cdot S + R$ in the CP decomposition, which results in a space complexity of $\mathcal{O}(R \cdot F \cdot S)$. The tensor train decomposition has $2 \cdot R \cdot F + (R^2 \cdot F) \cdot (S-2)$ parameters and a space complexity of $\mathcal{O}(R^2 \cdot F \cdot S)$. The Tucker decomposition still has $R \cdot F \cdot S + R^S$ parameters. Thus, it is only applicable for low S and when $R \ll F$.

3.2.2 Generalized Models

For training the tensor models, we take a maximum likelihood approach. Specifically, we minimize the negative log-likelihood of the model parameters given the data $\mathcal{D} = \{y^{(i)}, (x_1^{(i)}, \ldots, x_S^{(i)})\}_{i=1}^N$. For i.i.d. training data the negative log-likelihood is

$$l = -\sum_{i=1}^{N} \log p(y^{(i)} | x_1^{(i)}, \ldots, x_S^{(i)}, \Theta), \tag{3.5}$$

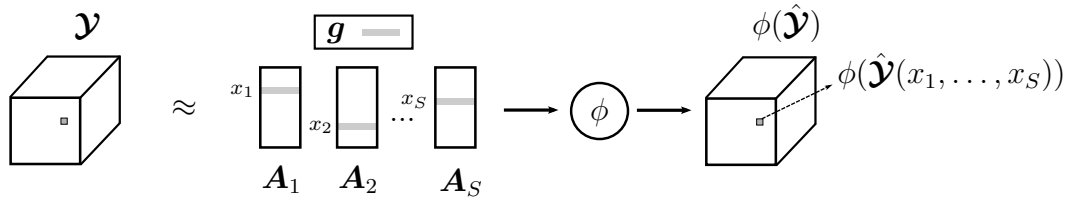


Figure 3.1: Predictive model using the CP decomposition. The tensors are plotted three dimensional for illustrative reasons ($S = 3$).

where Θ denotes the set of all parameters in the tensor decomposition. The probability is parametrized using the tensor decomposition models as

$$\mathbb{E}\left[p(y|x_1, \ldots, x_S)\right] = \phi(\hat{\mathcal{Y}}(x_1, \ldots, x_S)). \tag{3.6}$$

Similar to generalized linear models different distributions of the exponential family can be assumed for $p(y|x_1, \ldots, x_S)$, requiring different activation functions ϕ and leading to different cost functions. For each distribution within the exponential family the canonical link function, which maps from the expected value of the distribution to the natural parameter of the exponential family is uniquely determined. We apply the inverse canonical link function as an activation function ϕ to the model output, to map it to the range of the expected value of the distribution. Equation 3.5 can be minimized using gradient-based optimization algorithms. Note, that the cost function considers only non-zero elements of the tensor, i.e., the sparsity of the tensor is exploited. As Equation 3.5 sums over all training data points and not over the tensor elements, it can also handle situations where multiple training data points with the same input configuration but different target variables exist. In experiments, we found the stochastic optimization algorithm Adam [81] to work well for training the models. Figure 3.1 shows the CP decomposition of the sparse input tensor $\mathcal{Y}$. Predictions are derived by indexing the representations in the decompositions and applying the activation function ϕ. In the following we describe the activation functions and cost functions for different assumptions about the target variable.

Regression In regression tasks a real-valued Gaussian distributed target variable is assumed, such that

$$\boldsymbol{\mathcal{Y}}(x_1, \ldots, x_S) \sim \mathcal{N}(\hat{\boldsymbol{\mathcal{Y}}}(x_1, \ldots, x_S), \sigma^2). \tag{3.7}$$

The activation function for the Gaussian distribution is simply the identity function. σ^2 is typically assumed to be constant and independent of the input. In this case, the negative log-likelihood cost function becomes the mean squared error

$$l = \sum_{i=1}^{N} (\boldsymbol{\mathcal{Y}}(x_1^{(i)}, x_2^{(i)}, \ldots, x_S^{(i)}) - \hat{\boldsymbol{\mathcal{Y}}}(x_1^{(i)}, \ldots, x_S^{(i)}))^2. \tag{3.8}$$

The mean squared error cost function is most commonly used in the tensor decomposition literature. When fixing the parameters of all but one dimension, the tensor decompositions become linear functions, for which a closed form solution to the least squares problem exists. This allows to train the models using the alternating least squares algorithm. For the tensor train decomposition with mean squared error cost function, also a fast training algorithm based on Singular Value Decomposition has been proposed in [110]. Alternatively, the parameters can also be optimized using gradient-based methods.

Poisson Regression Poisson Regression is typically used when the target variable represents count data. We assume the target to be Poisson distributed, such that

$$\boldsymbol{\mathcal{Y}}(x_1, \ldots, x_S) \sim \text{Poisson}(\phi(\hat{\boldsymbol{\mathcal{Y}}}(x_1, \ldots, x_S))), \tag{3.9}$$

where the activation function ϕ for the Poisson distribution is $\exp(\cdot)$. The cost function for Poisson regression is

$$l = \sum_{i=1}^{N} \phi(\hat{\boldsymbol{\mathcal{Y}}}(x_1^{(i)}, \ldots, x_S^{(i)})) - \boldsymbol{\mathcal{Y}}(x_1^{(i)}, \ldots, x_S^{(i)}) \log \phi(\hat{\boldsymbol{\mathcal{Y}}}(x_1^{(i)}, \ldots, x_S^{(i)})). \tag{3.10}$$

Since there exists no closed form solution for this cost function, we propose to use gradient-based methods such as stochastic gradient descent (SGD) for the optimization of the model parameters.

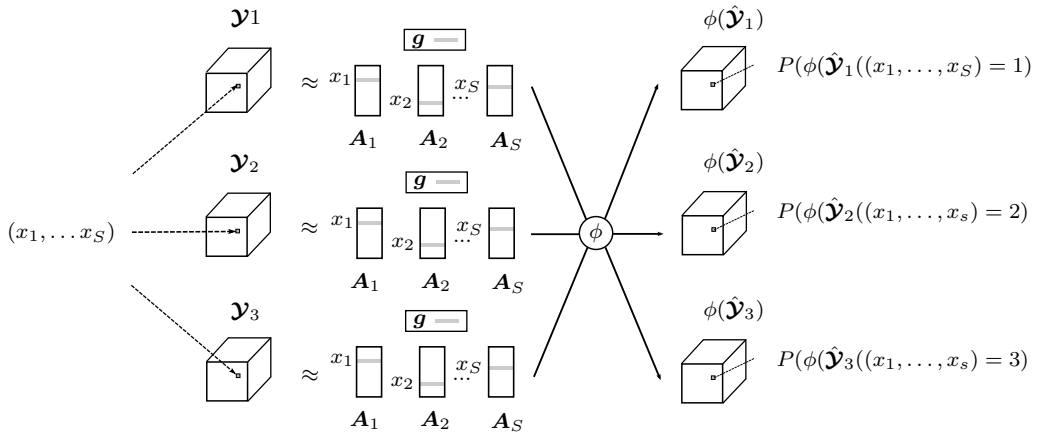


Figure 3.2: Multinomial classification model using CP decomposition. The tensors are plotted three dimensional for illustrative reasons ($S = 3$). The number of classes is also three in this example ($C = 3$).

Logistic Regression Logistic Regression is used for binary classifications problems, where the target variable is assumed to be Bernoulli distributed, such that

$$\boldsymbol{\mathcal{Y}}(x_1, \ldots, x_S) \sim \text{Bernoulli}(\phi(\hat{\boldsymbol{\mathcal{Y}}}(x_1, \ldots, x_S))), \tag{3.11}$$

with ϕ being the Sigmoid function

$$\phi(x) = \frac{1}{\exp(-x) + 1}. \tag{3.12}$$

The negative log-likelihood for Bernoulli distributed variables becomes the binary cross-entropy, which is

$$\begin{aligned} l = \sum_{i=1}^{N} & - \boldsymbol{\mathcal{Y}}(x_1^{(i)}, \ldots, x_S^{(i)}) \log(\phi(\hat{\boldsymbol{\mathcal{Y}}}(x_1^{(i)}, \ldots, x_S^{(i)})) \\ & - (1 - \boldsymbol{\mathcal{Y}}(x_1^{(i)}, \ldots, x_S^{(i)})) \log(1 - \phi(\hat{\boldsymbol{\mathcal{Y}}}(x_1^{(i)}, \ldots, x_S^{(i)}))). \end{aligned} \tag{3.13}$$

Also for this cost function no closed-form solution exists. Thus we optimize the cost function using a gradient-based method.

Multinomial Regression Multinomial regression is used to model multi-class classification problems. We assume a categorical distribution for the class label

$\boldsymbol{\mathcal{Y}}(x_1, \ldots, x_S)$ such that

$$\boldsymbol{\mathcal{Y}}(x_1, \ldots, x_S) \sim \text{Categorical}\big(\Big[\phi(\hat{\boldsymbol{\mathcal{Y}}}_1(x_1, \ldots, x_S)), \ldots, \phi(\hat{\boldsymbol{\mathcal{Y}}}_C(x_1, \ldots, x_S))\Big] \big), \tag{3.14}$$

where $\hat{\boldsymbol{\mathcal{Y}}}_c$ denotes a low rank tensor approximation of $\boldsymbol{\mathcal{Y}}_c$. This approach assumes one tensor decomposition for each class. The class tensors $\boldsymbol{\mathcal{Y}}_c$ are constructed by mapping all training examples that belong to the same class to a sparse tensor $\boldsymbol{\mathcal{Y}}_c \in \mathbb{R}^{F_1,\ldots,F_S}$ for $c \in \{1, ..., C\}$ where $C \in \mathbb{N}$ describes the number of classes. The tensors $\boldsymbol{\mathcal{Y}}_c$ are filled as

$$\boldsymbol{\mathcal{Y}}_c(x_1^{(i)}, \ldots, x_S^{(i)}) = \Pi(y^{(i)} = c), \quad \forall i \in \{1, ..., N\}, \tag{3.15}$$

where the indicator function $\Pi(x)$ is one if the statement x is true. Thus, in the class tensors all positions indexed by the training tuples are set to one, other positions are left unknown. Figure 3.2 shows the architecture of the multi-class model. Each class tensor is decomposed separately and the predictions are fed to the activation function ϕ, which in the case of a categorical distribution is the softmax function

$$\phi(\hat{\boldsymbol{\mathcal{Y}}}_c(x_1, \ldots, x_S)) = \frac{e^{\hat{\boldsymbol{\mathcal{Y}}}_c(x_1,\ldots,x_S)}}{\sum_k^C e^{\hat{\boldsymbol{\mathcal{Y}}}_k(x_1,\ldots,x_S)}}. \tag{3.16}$$

The softmax function normalizes the values across the low rank approximations such that $\sum_{c=1}^{C} \phi(\hat{\boldsymbol{\mathcal{Y}}}_c(x_1, \ldots, x_S)) = 1$. This makes the values interpretable as probabilities for the categorical distribution. The corresponding cost function for the categorical distribution is the categorical cross-entropy, which is defined as

$$l = -\sum_{i=1}^{N}\sum_{c=1}^{C} \Pi(y^{(i)} = c) \ \log \ \phi(\hat{\boldsymbol{\mathcal{Y}}}_c(x_1^{(i)}, \ldots, x_S^{(i)})). \tag{3.17}$$

All tensor decompositions are learned jointly end-to-end. Thus, optimal representations for the multinomial classification task are learned. When predicting a class label for a new input setting $(x_1, ..., x_S)$, which is not included in the training set, the representations for that tuple are indexed and the approximations $\hat{\boldsymbol{\mathcal{Y}}}_1$ to $\hat{\boldsymbol{\mathcal{Y}}}_C$ are computed. By applying the softmax function one obtains the class probabilities for that new input setting.

An alternative approach to modeling multi-class classification problems is using a one-versus-rest approach, where C binary classifiers are trained separately, each distinguishing one class from all other classes. To derive a prediction for a new data point, all classifiers are evaluated, and the class label of the classifier with the highest confidence is picked as a result.

3.2.3 Interpretability

In many use-cases the interpretability of a machine learning model, such as the contribution of a specific input variable to the prediction, is of interest. For example in a medical use case one might be interested in how the probability of having diabetes changes for a specific patient, if the blood pressure changes from medium to high. An advantage of linear models is that these insights can be derived directly from the learned coefficients. For non-linear models such as neural networks or support vector machines, it is computationally more involved to derive the desired insights. In this section, we analyze the interpretability of the multi-linear tensor models.

Odds Ratio For binary classification tasks an important interpretability measure is the odds ratio. We assume categorical input variables X_j for $j \in \{1, \dots S\}$, where each variable possibly takes discrete values $X_j \in \{1, \dots F_j\}$. For a given data sample $(x_1, \dots, x_S)$ the conditional odds ratio of changing variable X_j from x_j to any $x'_j \in \{1, \dots, F_j\}$, and leaving all other inputs unchanged, is defined as

$$\text{odds}(X_j; x_j \to x'_j) = \frac{p(Y=1|X_j=x_j, X_{k\neq j}=x_k)}{p(Y=1|X_j=x'_j, X_{k\neq j}=x_k)} \Big/ \frac{p(Y=0|X_j=x_j, X_{k\neq j}=x_k)}{p(Y=0|X_j=x'_j, X_{k\neq j}=x_k)}. \tag{3.18}$$

In logistic regression with discrete inputs, the effect of changing the value of a categorical variable from a discrete input to another can simply be derived from the model coefficients. For binary input variables $X_j \in \{0, 1\}$ the conditional odds ratio $\text{odds}(X_j; 0 \to 1)$ of changing this variable is $\exp(\beta_j)$, where β_j is the coefficient associated with X_j. For categorical input variables $X_j \in \{1 \dots F_j\}$, which are one-hot-encoded in the model input, the odds ratio can be derived as $\exp(\beta_{j,x_j} - \beta_{j,x'_j})$, where β_{j,x_j} and β_{j,x'_j} are the coefficients associated with the

positions of the one-hot-encoding corresponding to x_j and x'_j. In linear models the conditional odds ratio is independent of the remaining model input $X_{k\neq j}$. The time complexity for deriving the odds ratios for all possible changes in all input variables is $\mathcal{O}(F \cdot S)$ due to the computations of the differences of the coefficients and the computation of $\exp(\cdot)$.

Difference in Prediction A similar interpretability measure for regression tasks, is the absolute difference in the predicted value, when changing a categorical input.

For a given data sample $(x_1, \ldots, x_S)$ the conditional difference in the prediction, when changing variable X_j from x_j to any $x'_j \in \{1, \ldots, F_j\}$, and leaving all other inputs unchanged, is

$$\text{diff}(X_j; x_j \to x'_j) = \mathbb{E}\left[p(Y|X_j = x_j, X_{k\neq j} = x_k)\right] - \mathbb{E}\left[p(Y|X_j = x'_j, X_{k\neq j} = x_k)\right].$$

In the case of X_j being a binary input feature, the difference is simply β_j. For categorical variables which are encoded in a one-hot-representation the change can be calculated as $\beta_{j,x_j} - \beta_{j,x'_j}$, where β_{j,x_j} and β_{j,x'_j} are again the respective coefficients of the two different inputs in the one-hot-encoding. Similar to the odds ratio in logistic regression, the difference in the predicted value in linear regression, is independent of the remaining variables $X_{k\neq j}$. In the case of categorical inputs with a one-hot-representation, the time complexity for deriving the differences in the predicted value for all inputs is $\mathcal{O}(F \cdot S)$ due to the computations of the differences of the coefficients. For binary input variables the time complexity is $\mathcal{O}(1)$.

Non-linear models The direct interpretability of the model coefficients is a special case of linear models. For non-linear models, e.g., neural network or support vector machine, computing the conditional odds ratios for S input variables with F possible values requires $S \cdot F$ model evaluations. This leads to a complexity of $\mathcal{O}(F^2 \cdot S^2 \cdot H)$ for a MLP with one hidden layer of size H and to $\mathcal{O}(F^2 \cdot S^2 \cdot V)$ for a support vector machine with V support vectors. In contrast to the linear models, the conditional odds ratio in classification tasks, and the difference in the

Input: $(x_1, \ldots, x_S)$, $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_S$

$C_{\text{bw}} \leftarrow [\boldsymbol{A}_S(x_S)]$
for $i \leftarrow S-1, 1$ **do**
    $C_{\text{bw}}.\text{append}(\boldsymbol{A}_i(x_i) \odot C_{\text{bw}}.\text{last}())$

$C_{\text{fw}} \leftarrow \text{list}()$
$R \leftarrow \text{list}()$
for $i \leftarrow 1, S$ **do**
    if $i == 1$ **then**
        $R.\text{append}(\boldsymbol{A}_i \odot C_{\text{bw}}.\text{get}(i+1))$
        $C_{\text{fw}}.\text{append}(\boldsymbol{A}_i(x_i))$
    else if $i == S$ **then**
        $R.\text{append}(C_{\text{fw}}.\text{last}() \odot \boldsymbol{A}_i)$
    else
        $R.\text{append}(C_{\text{fw}}.\text{last}() \odot \boldsymbol{A}_i \odot C_{\text{bw}}.\text{get}(i+1))$
        $C_{\text{fw}}.\text{append}(C_{\text{fw}}.\text{last}() \odot \boldsymbol{A}_i(x_i))$

Output: R

Algorithm 3: Efficient computation of all possible changes in all input variables in the CP model. This computation is the basis for efficiently computing the conditional odds ratios.

predicted value in regression tasks, depends on the remaining unchanged inputs and is thus different for every input data point.

Tensor Decomposition Models For the CP decomposition and the tensor train decomposition, computing the conditional odds ratio or the conditional difference in the predicted value, can be performed in an efficient way. By substituting Equation 3.18 with the tensor decomposition models, with a sigmoid activation function, one derives

$$\text{odds}(X_j; x_j \rightarrow x_j') = \exp(\hat{\boldsymbol{\mathcal{Y}}}(X_j = x_j, X_{k \neq j} = x_k) - \hat{\boldsymbol{\mathcal{Y}}}(X_j = x_j', X_{k \neq j} = x_k)). \tag{3.19}$$

Input: $(x_1, \ldots, x_S)$, $\boldsymbol{A}_1, \boldsymbol{\mathcal{A}}_2, \ldots, \boldsymbol{\mathcal{A}}_{S-1}, \boldsymbol{A}_S$

$C_{\text{bw}} \leftarrow [\boldsymbol{A}_S(x_S)]$
for $i \leftarrow S-1, 1$ **do**
    $C_{\text{bw}}.\text{append}(\boldsymbol{\mathcal{A}}_i(x_i) \cdot C_{\text{bw}}.\text{last}())$

$C_{\text{fw}} \leftarrow \text{list}()$
$R \leftarrow \text{list}()$
for $i \leftarrow 1, S$ **do**
    if $i == 1$ **then**
        $R.\text{append}(\boldsymbol{A}_i \cdot C_{\text{bw}}.\text{get}(i+1))$
        $C_{\text{fw}}.\text{append}(\boldsymbol{A}_i(x_i))$
    else if $i == S$ **then**
        $R.\text{append}(C_{\text{fw}}.\text{last}() \cdot \boldsymbol{A}_i)$
    else
        $R.\text{append}(C_{\text{fw}}.\text{last}() \cdot \boldsymbol{\mathcal{A}}_i \cdot C_{\text{bw}}.\text{get}(i+1))$
        $C_{\text{fw}}.\text{append}(C_{\text{fw}}.\text{last}() \cdot \boldsymbol{\mathcal{A}}_i(x_i))$

Output: R

Algorithm 4: Efficient computation of all possible changes in all input variables in the tensor train model. This computation is the basis for efficiently computing the conditional odds ratios.

Thus, all possible odds ratios can be computed efficiently, if the model outputs $\hat{\boldsymbol{\mathcal{Y}}}$ for all changes in the input can be computed efficiently. Obviously, the same holds for the computation of the conditional difference in the predicted value in a regression model.

Algorithm 3 shows the procedure of computing the model outputs for all changes in the input for the CP decomposition. The input is a sample $(x_1, \ldots, x_S)$ and the model parameters $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_S$. The algorithm starts with a backward chain of multiplying the indexed representations based on the given input, where $\odot$ denotes the elementwise vector product. All intermediate results of the product chain are stored in the list C_{bw}. This loop has a time complexity of $\mathcal{O}(S \cdot R)$. After the backward loop follows a forward loop. The intermediate results are again stored in a list, which we denote C_{fw}. Additionally, at each step of the forward

Method	Complexity
Linear Model	$\mathcal{O}(F \cdot S)$
CP	$\mathcal{O}(F \cdot R \cdot S)$
tensor train	$\mathcal{O}(F \cdot R^2 \cdot S)$
Multilayer Perceptron	$\mathcal{O}(F^2 \cdot S^2 \cdot H)$
Support Vector Machine	$\mathcal{O}(F^2 \cdot S^2 \cdot V)$

Table 3.2: Runtime complexities of calculating all conditional odds ratios for S categorical inputs, with each having F different values.

chain, the outputs of changing the i-th input to any of the other possible values are computed and stored in the list R. For the computation of the outputs the intermediate results of the backward loop are utilized. The forward loop has a time complexity of $\mathcal{O}(F \cdot R \cdot S)$. Thus, computing the conditional odds ratios for all variables and a given input in the CP model has a time complexity of $\mathcal{O}(F \cdot R \cdot S)$.

Algorithm 4 shows the computation of the conditional odds ratios for the tensor train model. The basic procedure is the same as for the CP decomposition. It starts with a backward loop, where it performs the dot products between the representations and stores the intermediate results in C_{bw}. Due to the matrix representations in the tensor train, this results in a time complexity of $\mathcal{O}(S \cdot R^2)$. In every step of the backward loop, the output for all possible inputs is computed. The dot product in line 14 between the vector and the third order tensor is applied as a vector matrix product to each slice of the tensor. The second loop has a time complexity of $\mathcal{O}(F \cdot R^2 \cdot S)$, which is also the complexity of the overall algorithm.

Table 3.2 summarizes the time complexities for computing the conditional odds ratios of all inputs, for the different machine learning models. Logistic regression has the least complexity. Additionally, linear models have the advantage, that the conditional odds ratio is independent of the model input, and thus only needs to be computed once for all input data. The non-linear models have squared terms for F and S, which do not arise for the tensor models. Thus, the complexity of the tensor models is in-between the complexity of linear and non-linear models.

3.2.4 Mapping Continuous Inputs

The proposed model so far only works for a discrete input space. In the tensor decompositions, latent representations for each discrete input value are indexed and fused. In the CP decomposition, the representation at input i is a vector $\boldsymbol{a}_i = \boldsymbol{A}_i(x_i, :)$ depending on the discrete input variable x_i. One possible approach to modeling continuous inputs is to discretize the input space, and learn latent representations for each discretization step. However, this approach comes with the problem that it does not imply any smoothness on neighboring inputs. Although, this makes it a powerful, highly non-linear model, it is prone to overfitting. Thus, the model requires many more training examples to learn the smoothness implicitly. To introduce smoothness explicitly, and to extend the model to continuous inputs, we use smooth mappings from the input space to the latent parameters of the decomposition.

If we assume the discrete inputs to be represented in a one-hot-encoded vector, with $\boldsymbol{v}_i$ denoting the one-hot representation vector for x_i, the indexing of the latent representations can be written as a dot product between the representation matrix and the one-hot vector, such that

$$\boldsymbol{a}_i = \boldsymbol{A}_i \boldsymbol{v}_i. \tag{3.20}$$

This view gives rise to a natural extension to continuous input vectors $\boldsymbol{v}_i \in \mathbb{R}^d$; namely by learning a linear mapping from the input to the latent representation of the tensor decomposition. We can futher generalize this mapping to

$$\boldsymbol{a}_i = f(\boldsymbol{v}_i), \tag{3.21}$$

with $f(\cdot)$ being implemented by any, possibly non-linear, function approximator. Instead of indexing the latent representation from a matrix, their values are computed using the mapping function. To derive non-linear mappings we propose to use a Gaussian kernel, such that

$$\boldsymbol{a}_i(r) = \exp\left(-(\boldsymbol{\mu}_{i,r} - \boldsymbol{v}_i)^T \boldsymbol{D}_{i,r} \left(\boldsymbol{\mu}_{i,r} - \boldsymbol{v}_i\right)\right), \tag{3.22}$$

with $\boldsymbol{\mu}_{r_i} \in \mathbb{R}^d$ representing the centers and $\boldsymbol{D}_{r_i} \in \mathbb{R}^{d \times d}$ weighting the distance from the centers in the d dimensional input space. The closer a data point is to the center of a basis function, the higher is its activation. The centers of the basis functions can be seen as landmarks in the input space. In this way, the latent representation is modeled by the similarity of the input to the center of the Gaussian kernel function. Thus, similar inputs induce similar representations. The parameters of the basis function are optimized during training to yield optimal regression results. Also a mixture of discrete and continuous inputs can easily be modeled by applying the basis functions only to the continuous inputs, and learning representation matrices for the discrete input variables.

For the tensor train decomposition, where the latent representations are not vectors but matrices, the mapping can be applied in a similar manner, such that

$$\boldsymbol{A}_i(r_1, r_2) = \exp\left(-(\boldsymbol{\mu}_{i,r_1,r_2} - \boldsymbol{v}_i)^T \boldsymbol{D}_{i,r_1,r_2} (\boldsymbol{\mu}_{i,r_1,r_2} - \boldsymbol{v}_i)\right), \tag{3.23}$$

where $\boldsymbol{A}_i$ denotes the latent representation matrix of the i-th input and the scalars r_1, r_2 index their elements.

In the proposed model $\boldsymbol{v}_i$ is a vector of inputs. Thus, multiple input variables are grouped together to one dimension in the tensor decomposition. It is also possible to model every input by its own dimension in the tensor decomposition. However, in many use-cases, such as multi-view learning, a priori knowledge about groups of input variables is available, which can be exploited in the model design. The grouping of input variables reduces the dimensionality of the tensor decomposition, and thus the number of free parameters.

3.3 Related Work

Tensor models have been applied successfully in many application areas, such as relational learning, signal processing, spatio-temporal analysis, brain-wave analysis, and multilinear time invariant systems; see for example [106, 82, 131, 7, 112]. Most literature on tensor modeling is concerned with the decomposition of dense tensors, i.e., most of the elements in the tensor are non-zero. The factorization of sparse matrices has become popular in recommendation systems, especially due

to its success in the Netflix challenge; see [83]. The decompostion of sparse three-dimensional tensors found applications in the modeling of large knowledge bases, such as Yago [137], DBpedia [3], or Freebase [22]. In these models, the elements of the tensor represent all possible triple combinations of entities and relations in the knowledge graph. Only elements that represent known facts from the knowledge graph are set to one. These tensors are then decoposed by models such as RESCAL, in order to predict missing linkes in the knowledge graph; see [106]. Our approach builds uppon this line of research, by extending it to higher order tensors and applying it in new domains.

Our proposed model is strongly related to factorization machines [120, 121], which are polynomial models with factorized coefficients for the interaction terms. Additionally to the polynomial interaction terms, factorization machines also includes linear terms. In the multi-way factorization machine, which models all input interactions, the weights are factorized using the CP decomposition. However, for many applications, including recommendation settings, it has been shown sufficient, to model only pairwise interactions. In contrast to factorization machines, our proposed model also uses other decompositions, such as the tensor train decomposition. We further propose non-linear mapping functions for continuous inputs and the grouping of multiple inputs into one factorization dimension. Whereas, in factorization machines the mappings are linear and applied to each input variable separately.

Our model also shows some connections to sum-product networks [114]. Sum-product networks are a specific type of probabilistic graphical models, which are designed and trained in a way, such that they comprise generative models with tractable partition functions. If non-negative weights are assumed for tensor decompositions, the tensor decomposition function also forms a sum-product network. The non-negativity constraint in sum-product networks allows for efficient marginalization of the modeled joint probability. In our work, the tensor decompositions are trained in a discriminative way, which does not require marginalization operations. In our work, the structure of the compute network is naturally determined by the form of the tensor decomposition, whereas in sum-product networks the structure needs to be chosen based on the task, which can be a challenging design choice. Therefore, structure learning techniques have been proposed.

Sum-product networks have been applied to tasks such as image completion and language modeling; see [114, 31]. In [56], discriminative training of sum-product networks for image classification has been discussed.

Another use of tensor decompostion models in supervised machine learning is tensor regression. Tensor regression methods are linear regression models with input data, which is naturally structured in a multidimensional array. The weights of the tensor regression model are also represented in higher order tensors, which are compressed using low-rank tensor decompositions. Tensor regression allows for efficient modeling in settings where traditional methods are often insufficient due to the complex structure of the data and the high input dimensionality. As tensor regression learns a linear mapping and deals with dense input tensors, the approach is fundamentally different from ours; see [171, 174]. Stoudenmire et al. [136] proposed a tensor regression model which maps the data to a high-dimensional tensor using non-linear transformations, before applying a tensor regression model. The corresponding high dimensional weight tensor is then compressed using the tensor train decomposition.

Our approach using the non-linear mappings shows some similarities to RBF-networks which are able to approximate any non-linear function by using radial basis functions. RBF-networks have been successfully applied to a number of tasks including control engineering; see [26]. The main difference to our proposed functional tensor decomposition models is, that RBF-networks learn one latent representation for the entire input, and map it to the output; whereas, the continuous tensor decomposition models proposed in this work learn representations for each input mode and join them using the tensor decomposition model. In this way multi-way interactions are modeled explicitly.

3.4 Experiments

In this section, we evaluate the proposed tensor methods empirically. First, we present experiments on standard classification tasks with discrete input features. Second, we examine an application to modeling inverse dynamics for feedforward robot control. This shows the effectiveness of the tensor decomposition models when applied to continuous inputs.

3.4.1 Discrete Input Classification

Datasets For evaluating the performance of the proposed tensor model with discrete input variables, we apply the model to a number of standard classification data sets. We perform our experiments on five different datasets from the UCI data repository [40]. All datasets comprise classification tasks with categorical input features. These datasets naturally build a tensor, which contains the class for each possible input combination. However, during training only a subset of the elements in the tensor are known. The order of the tensor corresponds to the number of input variables. Table 3.3 summarizes the statistics of the five datasets.

The *Car* dataset contains features about cars, and the goal is to predict their acceptability in the market, which is encoded in four different output classes. The input consists of 6 categorical variables, where three of them can take four different values and the other three can take three different values. Thus, the total number of combinations is 1728. In the dataset, the classes for all possible input combinations are provided, which means that the tensor is completley filled. However, due to the split of training and test data, which is described in the next paragraph, the model is still trained only on a subset of the tensor elements.

In the *Nursery* dataset, applications to nursery schools are classified into 5 different classes. The input tensor has the size $2 \times 3 \times 3 \times 3 \times 3 \times 4 \times 4 \times 5$, which leads to 12960 possible configurations. Also in this dataset the full tensor is provided.

The *TickTacToe* dataset comprises a binary classification task. The inputs are the nine fields in the game, where each field can take one out of three different values. The values are *player1* or *player2*, if one of the two players occupied the field, or *blank* if the field is empty. The dataset contains all possible end configurations of the game. The goal is to predict, if player one has won the game.

The *Votes* dataset also contains a binary classification task. The goal is to predict the voting of congressmen, given 16 binary input variables. This leads to $2^{16} = 65536$ possible combinations. The dataset only contains 435 data points. Thus the input tensor is very sparse for this dataset.

In the *Connect-4* dataset, data samples consist of game states from the game connect-4. The game board is of size 6×7. Each field is described by a categorical

Dataset	# classes	# dimensions	# data points
Car	4	6	1728
Nursery	5	8	12960
TicTacToe	2	9	958
Votes	2	16	435
Connect-4	3	42	67557

Table 3.3: Meta information for the five classification datasets with categorical input features from UCI data repository [40]

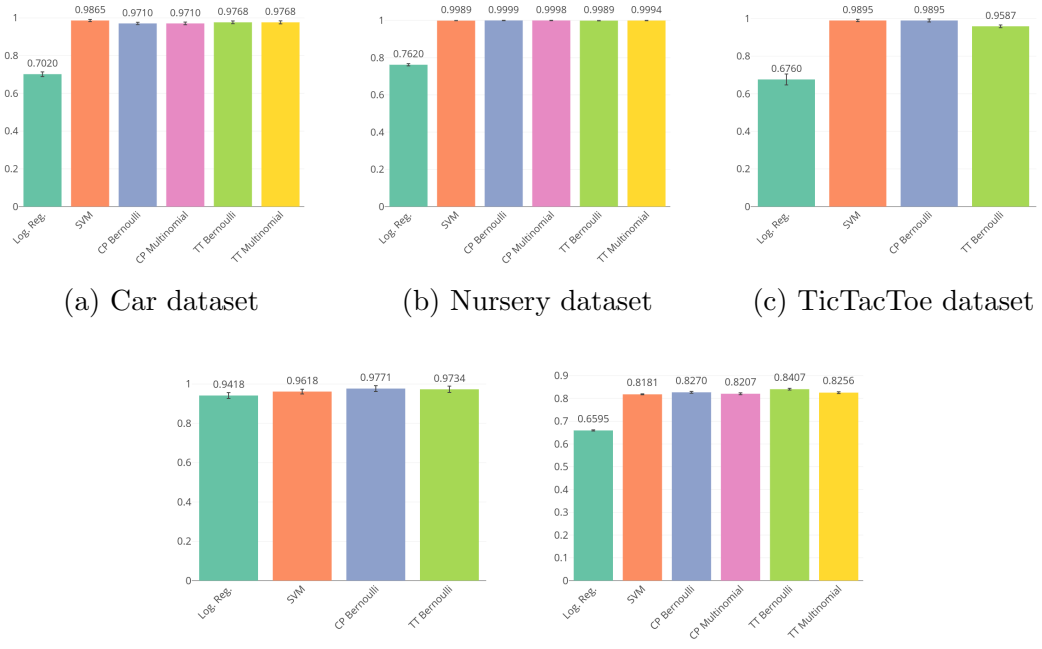
input variable, which can take three different values. The values are *player1* if a token of player one is placed on the field, *player2* if a token of player two is placed and *blank* if the field is not occupied. This setting leads to 3^{42} possible board configurations. However, not all configurations appear in practice, as the game ends as soon as one player has four tokens in a row and the game board can only be filled column-wise from bottom to top. The goal is to classify the outcome for player one, which can be either *win*, *loss* or *draw*. The whole dataset consists of 67557 game states, where none of the player has won yet and the next move is not forced.

Setting We split all datasets into 70 percent training and 30 percent test data. We use additional 5 percent of the training data for finding the best hyper parameters. The splits are randomly repeated 10 times. The task is to predict the target variable given the discrete inputs. We report the mean classification accuracy for all models along with their standard deviations.

We compare the classification results of the CP decomposition model and the tensor train decomposition model against state-of-the-art linear and non-linear machine learning models, namely, logistic regression and support vector machine with a Gaussian kernel. For these two models the input data is encoded in a concatentation of one-hot feature vectors; one for each input variable. Logistic regression is regularized using the L2 norm. The amount of regularization and the penalty term for the support vector machine where tuned on the validation set. We use the implementations from the Python package scikit-learn [27] for the baseline models.

We compare the baselines with the CP and the tensor train decomposition approach described in this chapter. As the input tensors for the different datasets have between 6 and 42 dimensions, the full Tucker decomposition is not tractable for any of the datasets. For the datasets where the prediction task is a multi-class classification problem we use two different approaches. The first is a one-versus-rest approach, where a binary classifier is trained for each class, to distinguish data points from this class from data points from other classes. In the prediction the class label of the classifier with the highest confidence is picked as a result. The second approach is using the multinomial classification as described in Section 3.2.2 and Equation 3.14, with multiple class tensors which are trained jointly and normalized using the softmax function. We initialized the weights of the CP decomposition uniformly between 0.9 and 1.1. The representation matrices of the tensor train decomposition where initialized by the identity matrix. The optimal rank of the tensor decomposition models was determined on the validation set. For optimizing the tensor decomposition, we applied the Adam optimization method [81] which has one hyperparameter, namely the initial learning rate. We evaluate the performance of the model on the validation set after each epoch, and stopped training, when the accuracy has not been increased within the last 10 epochs (early stopping). The tensor decompositions where implemented using the Python package Theano [143] and Keras [35].

Results Figure 3.3 summarizes the results of the experiments. In all datasets the support vector machine shows significantly better results than the logistic regression model. This shows that there is a strong non-linear relationship between the input variables and the class labels. In all five datasets the tensor decompositions reach about the same accuracy as the support vector machine. For the multiclass tasks, it does not make a significant difference if the model is trained using a one-versus-rest approach or the multinomial regression approach. Also, in most cases the tensor train decomposition and the CP decomposition result in very similar accuracy. In the TicTacToe dataset the tensor train decomposition performs even worse than the CP decomposition. Thus, the additional parameters of the tensor train decomposition have not shown to be advantageous for the predictive modeling in these experiments.



(a) Car dataset (b) Nursery dataset (c) TicTacToe dataset

(d) Votes dataset (e) Connect-4 dataset

Figure 3.3: Mean classification accuracy and standard deviation on the five different datasets from the UCI data repository [40]

For the CP decomposition a rank of 10 has been found sufficient in all five datasets. The optimal tensor train rank was determined at 10 for the Connect-4 dataset, and 5 for all other datasets. The optimal rank for the Bernoulli models was the same as for the multinomial models. The best initial learning rate for the Adam optimizer was found at 0.0001 for the tensor train decompositions, and at 0.01 for the CP decompositions in all experiments. We found that limiting the rank of the tensor decomposition models and early stopping was sufficient for avoiding overfitting in the tensor decomposition models. It was not necessary to apply additional regularization to the parameters.

The L2 regularization for the linear models has been found optimal at an inverse strength of 0.01 for the car and nursery dataset. For TicTacToe and Connect-4 0.1 was found optimal and for the votes dataset 1.0 has been determined. The optimal penalty terms for the support vector machine have been found to be 100 for all datasets except the Votes datasets, there a penalty term of 1 has been found

Method	Complexity
Logistic Regression	$\mathcal{O}(S)$
Support Vector Machine	$\mathcal{O}(S \cdot V)$
CP	$\mathcal{O}(S \cdot R)$
tensor train	$\mathcal{O}(S \cdot R^2)$

Table 3.4: Comparison of the runtime complexities of the different models.

to perform optimal. These high penalty terms led to a large amount of support vectors. In the connect-4 dataset the number of support vectors for the first class are 95 percent of the training data. In all other datasets, the number of support vectors is also very high with 50 to 75 percent. Only in the nursery dataset the number of support vectors is relatively small between 1 and 14 percent depending on the class.

Table 3.4 compares the complexity of the different models. The logistic regression has a linear complexity in the number of input features, however, its modeling capabilities are also limited to linear separating hyperplanes. For all models we dropped the term F which describes the number of options for the categorical inputs. As for a given data point only one value out of F is active at each input, deriving the respective weight or the latent representation, can be implemented in $\mathcal{O}(1)$, by indexing an array. In the comparison we assume an efficient implementation, which exploits this sparsity, for all models. Support vector machines have a complexity of $\mathcal{O}(S \cdot V)$, where V are the number of support vectors. If the support vector machine degenerates to having a large amount of support vectors, the complexity of the model is dominated by this term. In this case the tensor decompositions show a clear advantage. The complexity of the CP decomposition is $\mathcal{O}(S \cdot R)$ where R is the rank of the decomposition, which has been found to be much less than the number of support vectors. Even in the tensor train decomposition, where the rank appears squared in the complexity, the term is small compared to the large number of support vectors.

3.4.2 Application to Inverse Dynamics

In this section, we evaluate the extension of the tensor decomposition models to continuous inputs on the problem of learning inverse dynamics. Within model-based robot control, an inverse dynamics model is used to compute the necessary joint torques of the robot's motors for the execution of a desired movement. The feedforward control command can be calculated using the rigid-body formulation $\boldsymbol{u}_{FF} = M(\boldsymbol{p})\ddot{\boldsymbol{p}} + F(\boldsymbol{p}, \dot{\boldsymbol{p}})$, with $\boldsymbol{p}, \dot{\boldsymbol{p}}, \ddot{\boldsymbol{p}}$ being vectors of joint positions, joint velocities, and joint accelerations. However, in practice many nonlinearities such as friction or actuator forces need to be taken into account. Thus, methods modeling $\boldsymbol{u}_{FF} = f(\boldsymbol{p}, \dot{\boldsymbol{p}}, \ddot{\boldsymbol{p}})$ using non-linear regression techniques have shown superior performance in inferring the required joint torques for feedforward robot control. The parameters of the function f are estimated offline using collected trajectories of the robot [38, 104, 99]. An additional feedback component is typically used to prevent the accumulation of tracking errors.

In this section, we build upon the approach of decomposing sparse tensors and apply it to inverse system identification. Our model exploits the inherent three-way interaction of *positions* $\times$ *velocities* $\times$ *accelerations*. We derive a continuous version of tensor decompositions by mapping the continuous inputs using basis functions, specifically Gaussian kernels. The basis functions also imply smoothness on the inputs, such that the model is able to generalize well, in spite of the extreme sparsity. By using multivariate basis functions, we can group inputs such that the dimensionality of the tensor decomposition can be reduced. In our inverse dynamics model, we group the joint positions, velocities, and accelerations of all degrees of freedom of the robot, resulting in a tensor of order three. This makes the powerful Tucker decomposition applicable to the problem. We evaluate our proposed method on an inverse dynamics dataset including movements from a seven degrees of freedom SARCOS robot arm. We compare against other state-of-the-art regression techniques for this task.

Inverse dynamics are traditionally modeled using the rigid-body formulation, see [38]. However, general regression techniques such as locally weighted projection regression (LWPR), Gaussian processes, and RBF-networks, have shown advantageous for learning inverse dynamics; see [157, 118]. The topic was subject to

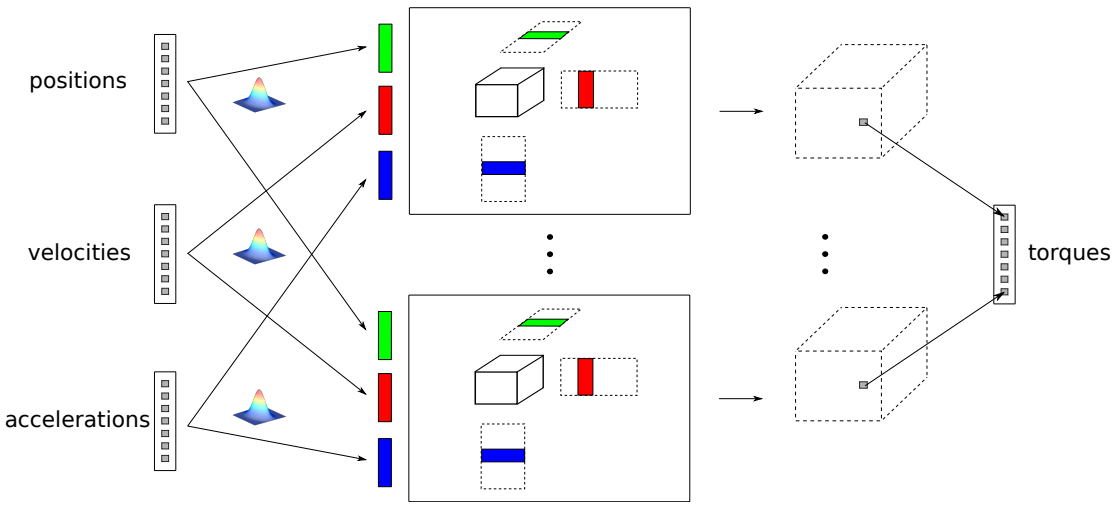


Figure 3.4: Inverse dynamics model using a functional Tucker decomposition. The output tensors and the representation matrices are replaced by functions (illustrated with dashed lines). The representations are computed given the continuous inputs using Gaussian kernels.

a number of studies; see [28, 104]. Support vector regression has shown superior performance for this task.

Continuous Tensor Model We describe a continous Tucker model for the approximation of the joint torques, necessary to perform a movement of a robot arm. Figure 3.4 shows the model schematically. We consider a robot with $C \in \mathbb{N}$ degrees of freedom (DoF). In the following, we denote the vectors $\boldsymbol{p}, \dot{\boldsymbol{p}}, \ddot{\boldsymbol{p}}$, describing the desired positions, velocities, and accelerations for each of the C DoFs, as the input variables $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3 \in \mathbb{R}^C$. The vector $\boldsymbol{y} \in \mathbb{R}^C$ describes the corresponding joint torques. Each element of the vector $\boldsymbol{y}$ is modeled in a separate function. We model the c-th joint torque $\boldsymbol{y}(c) = f_c(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$ for $c \in \{1, \ldots, C\}$ using functional tensor decomposition functions. Each input vector is modeled by one dimension in a third order Tucker decomposition, which describe the joint torques. The Tucker decomposition models the three-way interaction *positions* $\times$ *velocities*

$\times$ *accelerations* with a limited rank R in each dimension, such that

$$f_c(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3) = \sum_{r_1,r_2,r_3}^{R} \boldsymbol{\mathcal{G}}_c(r_1, r_2, r_3) \cdot A_1(\boldsymbol{x}_1, r_1) \cdot A_2(\boldsymbol{x}_2, r_2) \cdot A_3(\boldsymbol{x}_3, r_3). \tag{3.24}$$

A_1 to A_3 are functions, which map from the c-dimensional input to the latent representations of the Tucker model. We model the representations using multivariate Gaussian kernels, such that

$$A_i(\boldsymbol{x}_i, r_i) = \exp\left(-(\boldsymbol{\mu}_{r_i} - \boldsymbol{x}_i)^T \boldsymbol{D}_{r_i} (\boldsymbol{\mu}_{r_i} - \boldsymbol{x}_i)\right) \quad \forall i \in \{1, 2, 3\}, \tag{3.25}$$

with $\boldsymbol{\mu}_{r_i} \in \mathbb{R}^C$ representing the centers and $\boldsymbol{D}_{r_i} \in \mathbb{R}^{C \times C}$ weighting the distance from the centers in the C dimensional input space. The closer a data point is to the center of a basis function, the higher is its activation. Thus, the centers of the basis functions can be seen as landmarks in the input space. All three-way interactions, between the representations of the three input dimensions, are explicitly modeled and weighted by the elements of the core tensor $\boldsymbol{\mathcal{G}}_c$.

As discussed in Section 3.2.2 we train the model, taking a maximum likelihood approach. As we deal with a regression task we apply the mean squared error cost function on the decompositions. We minimize Equation 3.8 using gradient descent. In experiments, we found the stochastic optimization algorithm Adam [81], which adopts the learning rate automatically for each parameter, to work best for this task. The sampling of stochastic mini-batches for each update has also shown advantageous for speeding up training.

We initialize the centers of the Gaussian kernel in a preprocessing step, using three k-means clusterings [89], such that

$$J_i = \sum_{r=1}^{R} \sum_{j=1}^{N} \|\boldsymbol{x}_i^{(j)} - \boldsymbol{\mu}_{r_i}\|^2 \tag{3.26}$$

are minimized for $i \in \{1, \ldots, 3\}$. All matrices $\boldsymbol{D}_{r_i}$ are initialized with the identity matrix. The elements of the core tensors $\boldsymbol{\mathcal{G}}_c$ are initialized randomly with

a Gaussian distribution of mean zero and standard deviation 0.05. While training all parameters are further optimized. We implemented the model using the computational Python library Theano [143] and Keras [35].

Dataset The dataset was introduced by [157].[1] It contains data from a SARCOS robot arm with seven degrees of freedom. The data was collected from the moving robot arm at 100Hz and corresponds to 7.5 minutes of movement. The dataset includes 21 input dimensions, consisting of 7 joint torques, 7 joint positions, 7 joint velocities, and 7 joint accelerations. The whole dataset consists of 42482 samples. We split the dataset randomly into 90 percent training and 10 percent test data. Additional 5 percent of the training set where used as a validation set. The task is to learn a model on the training data, which models the 7 joint torques, given the positions, velocities and accelerations. The offline learned model can then be applied in the forward controller of the robot. The dataset has been subject to some studies on the topic; see [157, 118]. The regression task has been found to be highly non-linear. Non-linear regression techniques outperformed the rigid-body dynamics formulation by a large margin. The performance of the regression techniques is evaluated on the test set, which includes unseen movements. We repeated the random split 10 times and report the average results and the standard deviation of multiple trials.

Baseline Methods We compare our model against various state-of-the art regression techniques, modeling the function $\boldsymbol{y} = f(\boldsymbol{p}, \dot{\boldsymbol{p}}, \ddot{\boldsymbol{p}})$. The baseline models we consider are linear regression, RBF-networks and support vector regression. In previous studies support vector regression has shown the best results on this task. For all baseline models a concatenated vector $\boldsymbol{x} = [\boldsymbol{p}, \dot{\boldsymbol{p}}, \ddot{\boldsymbol{p}}]$ is built. The linear regression model learns a linear mapping from the inputs to the outputs, such that

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}. \tag{3.27}$$

[1] http://www.gaussianprocess.org/gpml/data/

Method	DoF 1	DoF 2	DoF 3	DoF 4	DoF 5	DoF 6	DoF 7
LR	6.80	11.62	10.82	5.81	12.81	22.59	6.73
RBF-Net	2.64	1.79	1.01	0.41	4.07	3.91	1.17
SVR	0.88	0.67	0.43	0.15	1.04	0.72	0.34
RBF-Tucker	0.59	0.28	0.46	0.24	1.03	0.91	0.31
RBF-CP	1.64	1.14	0.61	0.32	1.30	1.17	0.50

Table 3.5: Normalized mean squared error for all 7 degrees of freedom in percent.

RBF-networks model the regression problem as

$$\boldsymbol{y}(c) = \sum_{i=1}^{R} w_{i,c} \exp\left(-\beta_i \|\boldsymbol{x} - \boldsymbol{c}_i\|^2\right) + b_c. \tag{3.28}$$

The parameters $\boldsymbol{c}_i$, β_i, $w_{i,c}$, and b_c are learned using backpropagation. We initialized the parameters $\boldsymbol{c}_i$ with the centroids of a k-means clustering on the training data, where R is the number of centroids.

Support vector regression [134] has shown state-of-the-art results in modeling inverse dynamics. It predicts $\boldsymbol{y}$ as

$$\boldsymbol{y}(c) = \sum_{j=1}^{N} (\alpha_{j,c} - \alpha_{j,c}^{\star}) k_c(\boldsymbol{x}^{(j)}, \boldsymbol{x}) + b_c, \tag{3.29}$$

with $k(\boldsymbol{x}, \boldsymbol{x}')$ being a kernel function. In the experiments we use a Gaussian kernel. α_j and $\alpha_j^{\star}$ are Lagrange multipliers, which are determined during optimization. In our experiments we use the Python library scikit-learn [27].

Furthermore, we compare the functional Tucker model with a functional CP model. For the functional CP model we replace the tensor decomposition structure in Equation 3.24 with a CP decomposition, as shown in Equation 3.2.

Results We report the normalized mean squared error (nMSE) for the regression task, which is defined as the mean squared error of all data points divided by the variance of the target variable in the training data. Table 3.5 summarizes the mean nMSE for all seven degrees of freedom in percent. In Table 3.6, the mean of all seven degrees of freedom is shown. All results, as well as the standard deviation are

Method	Mean ± std in %
LR	11.03 ± 0.26
RBF-Net	2.14 ± 0.19
SVR	0.60 ± 0.28
RBF-Tucker	0.55 ± 0.24
RBF-CP	0.96 ± 0.22

Table 3.6: Normalized mean squared error in average for all 7 degrees of freedom in percent. Mean and standard deviation of ten random data splits.

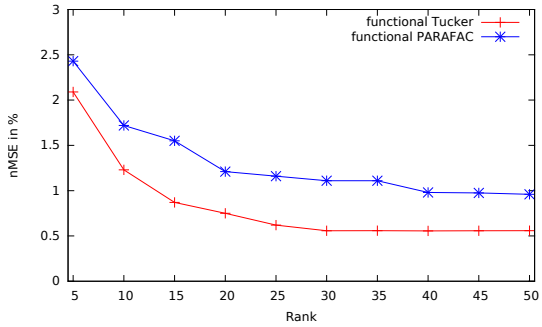


Figure 3.5: Normalized mean squared error of the Tucker and CP model, in dependency of the embedding rank.

referring to the average result of 10 random data splits. The performance of the regression techniques varies across the DoFs. The linear model reaches an nMSE of 11.03% in average. The nonlinear RBF-network performs much better with an nMSE of 2.14% in average. The number of of hidden neurons for the RBF-network was set to 1000. With larger numbers the predictive performance did not increase. The support vector regression model yields a very good result of 0.60%. Here, we set the parameter C to 600 and ϵ to 0.1. All hyperparameters were evaluated on a separate validation set. Our proposed continuous Tucker model resulted in a slightly better nMSE of 0.55%. Especially, for the first two DoFs the continuous Tucker model performs significantly better than support vector regression. For the other DoFs the results of support vector regression and continuous Tucker decomposition are very close to each other. The parameter efficient continuous CP model reaches an nMSE of 0.96% in average. Figure 3.5 shows the performance of the two continuous tensor decomposition models in dependence of the rank of

Method	training time	prediction time
SVR	$\sim$ 5.1 h	$\sim$ 3.0 ms
RBF-Tucker	$\sim$ 1.9 h	$\sim$ 0.7 ms

Table 3.7: Computation time for one DoF.

the decompositions. For the Tucker model, the performance converges at a rank of 30 and for the CP model at a rank of 40. It is also notable that both methods already perform relatively well with a very small rank of 5. The nMSE of the Tucker model is 2.09% with a rank of 5 and the nMSE of the CP model is 2.43%. Both continuous tensor models show clearly better results than RBF-networks. This indicates that the explicit modeling of the three-way interaction, yields a significant improvement.

Table 3.7 shows the training and prediction time of support vector regression and the continuous Tucker model for one DoF on a single core of a Intel Core I5-4300M 2.6 GHz CPU. The training time until convergence is more than 2.5 times faster for the continuous Tucker model, and the prediction of a single datapoint is more than 4 times faster. Especially the speedup in prediction is important in order to apply the algorithm at a high sampling rate to real time control. The high computational cost of the SVR is caused by the fact that more than 90 percent of the training data points are support vectors.

3.5 Conclusion

We have shown how tensor decompositions can be applied to supervised machine learning by mapping the data to a sparse and high-dimensional tensor. This approach can be easily applied to regression and classification tasks. Any tensor decomposition can be used for the model, as long as it scales to the order of the tensor. As latent representations are learned for all possible values of the input variables, the approximated function can be highly non-linear. We discussed the runtime and space complexity, which turned out to be advantageous over conventional non-linear machine learning models. We also presented an algorithm for efficiently deriving interpretability measures, such as the odds ratio. In ad-

dition, we discussed the modeling of differently distributed target variables, such as Gaussian, Bernoulli, Poisson, and the categorical distribution. To obtain a categorical distribution over the class labels describing the overall system state, multiple class tensors are jointly decomposed, and representations between the decompositions can be shared for some decompositions. Our experiments on a number of classification datasets with discrete input variables have shown that the tensor decompositions reach similar performances as support vector machines.

Furthermore, we proposed to augment the tensor decompositions with basis functions so as to allow for continuous input variables. In this way, a continuous version of a tensor decomposition can be derived. Representations for each tensor mode are induced through the basis functions and fused by the tensor model. The parameters of the basis functions can be learned through backpropagation. In the experiments section, we applied a tensor model based on the Tucker decomposition to the modeling of inverse dynamics. Our proposed model exploits the inherent three-way interaction of *positions* $\times$ *velocities* $\times$ *accelerations*. The grouping of multiple input variables into one tensor dimension makes the powerful Tucker model applicable. Experiments on an inverse dynamics dataset, derived from a seven degrees of freedom robot arm, show promising results from our proposed model for the application of learning inverse dynamics. The proposed continuous Tucker model outperforms RBF-networks, and even support vector regression, which has shown state-of-the-art performance on this task.

Our extension of tensor decomposition models to continuous inputs enables a wide range of applications. In particular, if an inherent multi-way structure exists in the data, continuous tensor models can be advantageous over traditional techniques by explicitly modeling the multi-way interaction. The experimental results on both datasets show the effectiveness of our approach and indicate the great potential of sparse tensor models for supervised machine learning.

Chapter 4

Visual Relationship Detection with Learned Semantic Models

In this chapter, we consider the multi-modal problem of fusing semantic and sensory information for the task of visual relationship detection. We propose two models, both combining tensor decompositions and object detection in different ways. Experiments on a recently released dataset for visual relationship detection show promising results for this novel direction of research.

The contributions of this chapter are published in:

[10] Stephan Baier, Yunpu Ma, and Volker Tresp. Improving visual relationship detection using semantic modeling of scene descriptions. *International Semantic Web Conference*, 2017

[11] Stephan Baier, Yunpu Ma, and Volker Tresp. Improving information extraction from images with learned semantic models. *International Joint Conference on Artificial Intelligence*, 2018

Section 4.1, 4.2, 4.3 are taken from [10] with some edits and extensions. Section 4.3.3 has been taken and extended from Section 3.2 in [11]. Sections 4.4, 4.5, and 4.6 have been partly taken and edited from the Sections 2, 4, and 5 in [10]. All figures from this chapter, have been published with minor modifications in [10]. I am the main author of the papers [10] and [11]. The papers have been written by me, and all the experiments have been conducted by me.

4.1 Introduction

Extracting semantic information from unstructured data, such as images or text, is a key challenge in artificial intelligence. Semantic knowledge in a machine-readable form is crucial for many applications, such as search, semantic querying, and question answering.

Novel computer vision algorithms, mostly based on convolutional neural networks (CNN), have enormously advanced over the last years. Standard applications are image classification and, more recently, also the detection of objects in images. However, the semantic expressiveness of image descriptions that consist simply of a set of objects is rather limited. Semantics is captured in more meaningful ways by the relationships between objects. In particular, visual relationships can be represented by triples of the form (*subject*, *predicate*, *object*), where two entities appearing in an image are linked through a relation (e.g. *man-riding-elephant*, *man-wearing-hat*). Figure 4.1 illustrates the task of visual relationship detection on an example. The input data is a raw image, and the output are a number of detected objects plus their relationship described as semantic triples.

Extracting triples, i.e. visual relationships, from raw images is a challenging task, which has been a focus in the Semantic Web community for some time [5, 21, 16, 152, 170, 128] and recently also gained substantial attention in main stream computer vision [126, 34, 30, 91]. First approaches used a single classifier, which takes an image as input and outputs a complete triple [126, 34]. However, these approaches do not scale to datasets with many object types and relationships. Due to the cubic combinatorial complexity of possible triples it is likely that not all relevant triples do appear in the training data, which makes training a predictive model difficult. Recently, [91] proposed a method which classifies the visual objects and their relationships in independent preprocessing steps, and then derives a prediction score for the entire triple. This approach was applied to the extraction of triples from a large number of potential triples. In the same paper, the first large-scale dataset for visual relationship extraction was published.

The statistical modeling of graph-structured knowledge bases, often referred to as knowledge graphs, has recently gained growing interest. The most popular approaches learn embeddings for the entities and relations in the knowledge graph,

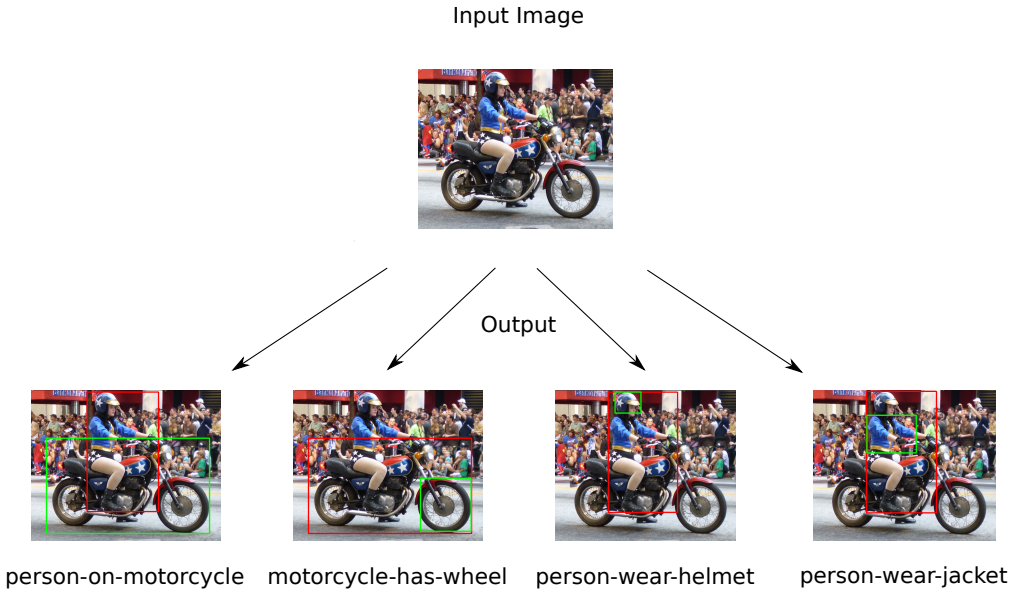


Figure 4.1: Input to the model is a raw image. The task of visual relationship detection is to detect the objects in the image and generate semantic triples, which describe the scene. Images taken from the Stanford Visual Relationship dataset [91].

by decomposing the three-dimensional adjacency tensor of the knowledge graph with limited rank. Based on the learned representations from the tensor decomposition, a likelihood for every possible triple can be derived. This approach has mainly been used for link prediction, which tries to predict missing triples in a knowledge graph. A recent review paper can be found in [106].

In the approaches described in this chapter, statistical knowledge base models, based on tensor decompositions, are used to support the task of visual relationship detection. The semantic model helps to interpret a visual scene. For example if the visual model detects a motorbike, it is very likely that the triple *motorbike-has_part-wheel* is true, as all motorbikes have wheels. We integrate such prior knowledge by fusing the visual models with learned semantic models. In particular, we propose two different approaches, both building on probabilistic semantic machine learning models.

The first model uses a Bayesian fusion approach for combining visual object detection methods with a separate trained probabilistic semantic prior. Incorpo-

rating a probabilistic semantic prior especially helps in cases where the prediction of the classifier is not very certain, and for the generalization to unobserved triples in the training set. A purely visual model, which only processes the raw image data, is combined with a semantic prior. For combining the semantic prior with the visual model we employ a probabilistic approach which can be divided into a semantic part and a visual part. We show how the semantic part of the probabilistic model can be implemented using standard link prediction methods using tensor decompositions, and the visual part using recently developed computer vision algorithms. We train our semantic model by using absolute frequencies from the training data, describing how often a triple appears in the training data. By applying a latent variable model, we are able to also generalize to unseen or rarely seen triples, which still have a high likelihood of being true, due to their similarity to other likely triples. For example if we frequently observe the triple *person-ride-motorcycle* in the training data we can generalize also to a high likelihood for *person-ride-bike* due to the similarity between *motorcycle* and *bike*, even if the triple *person-ride-bike* has not been observed or just rarely been observed in the training data. The similarity of *motorcycle* and *bike* can be derived from other triples, which describe, for example, that both have a *wheel* and both have a *handlebar*.

The second model we propose, is a conditional multi-way model which is inspired by statistical link prediction methods. This model does not include an explicitly trained prior of the semantic triples. The prior is rather captured in the learned entity representations of a multi-way neural network, which is applied subsequently to the output of the visual model. Thus, the conditional multi-way model can be trained in a purely feedforward manner. Similar to the first model, we also reach a generalization to unobserved or rarely observed triples. This is achieved by learning latent representations for all involved entities in the conditional multi-way model. Given the latent representations for a pair of visual objects which have been detected in the image, we predict the relationship among them using the conditional multi-way model. As similar entities based on their occurrence in the training data should get similar latent representations, this helps to generalize to triples which are reasonable, but have never been observed in the training data.

We conduct experiments on the Stanford Visual Relationship dataset recently published by [91]. We evaluate the models on the task of predicting semantic triples and the corresponding bounding boxes of the subject and object entities detected in the image. For the Bayesian fusion model, we compare various tensor decomposition models as a semantic prior. Our experiments show, that including the semantic model improves on the state-of-the-art result in the task of mapping images to their associated triples. The experiments further show that the conditional multi-way model, especially in the task of predicting unobserved triples, achieves performance that is comparable to the Bayesian fusion model.

This chapter is structured as follows. Section 4.2 gives an overview over the most important background for the chapter. In particular, we discuss various tensor decomposition models for link prediction and approaches to image classification and object detection. In Section 4.3, we describe the Bayesian fusion model and the conditional multi-way model. Section 4.5 contains the experimental evaluation. Finally, we conclude our work with Section 4.6.

4.2 Background Methods

Our proposed models join ideas from two areas, computer vision and statistical relational learning for semantic modeling. Both fields have developed rapidly in recent years. In this chapter, we discuss relevant work from both areas.

4.2.1 Statistical Link Prediction

A number of statistical models have been proposed for modeling graph-structured knowledge bases often referred to as knowledge graphs. Most methods are designed for predicting missing links in the knowledge graph. A recent review on link prediction can be found in [106]. A knowledge graph $\mathcal{G}$ consists of a set of triples $\mathcal{G} = \{(s, p, o)_i\}_{i=1}^{N} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$. The entities $s, o \in \mathcal{E}$ are referred to as *subject* and *object* of the triple, and the relation between the entities $p \in \mathcal{R}$ is referred to as *predicate* of the triple.

Most popular link prediction methods can be seen as tensor decompositions on the adjacency tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{R}|\times|\mathcal{E}|}$ of the graph $\mathcal{G}$. The adjacency tensor is

filled as

$$\boldsymbol{\mathcal{X}}(s,p,o) = \Pi((s,p,o) \in \mathcal{G}) \qquad \forall \;\; s,o \in \{1,\ldots,|\mathcal{E}|\} \;\; p \in \{1,\ldots,|\mathcal{R}|\} \tag{4.1}$$

where $\Pi(\cdot)$ is one, if the condition $((s,p,o) \in \mathcal{G})$ is true. The adjacency tensor $\boldsymbol{\mathcal{X}}$ is approximated by $\hat{\boldsymbol{\mathcal{X}}} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{R}|\times|\mathcal{E}|}$ which is derived through a decomposition of $\boldsymbol{\mathcal{X}}$ with limited rank. In the following, we describe the decompositions, which are used in this chapter.

DistMult DistMult [166] scores a triple by building the tri-linear dot product of the embeddings, such that

$$\boldsymbol{\mathcal{X}}(s,p,o) \approx \langle \boldsymbol{A}(s,:), \boldsymbol{R}(p,:), \boldsymbol{A}(o,:)\rangle = \sum_{r=1}^{\tilde{r}} \boldsymbol{A}(s,r)\boldsymbol{R}(p,r)\boldsymbol{A}(o,r), \tag{4.2}$$

with $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{E}|\times\tilde{r}}$ containing the latent representations for all entities and $\boldsymbol{R} \in \mathbb{R}^{|\mathcal{R}|\times\tilde{r}}$ containing the latent representations for all relations. The dimensionality $\tilde{r}$ of the embeddings, also called rank, is a hyperparameter of the model. Note, that this model is a CP decomposition, with shared factor matrices for the first and third dimension. Thus, DistMult is not able to model non-symmetric relations.

ComplEx ComplEx [147] extends DistMult to complex valued vectors for the embeddings of both, relations and entities. The decomposition is

$$\begin{aligned}
\boldsymbol{\mathcal{X}}(s,p,o) &\approx Re(\langle \boldsymbol{A}(s,:), \boldsymbol{R}(p,:), \overline{\boldsymbol{A}(o,:)}\rangle) \\
&= \langle Re(\boldsymbol{A}(s,:)), Re(\boldsymbol{R}(p,:)), Re(\boldsymbol{A}(o,:))\rangle \\
&+ \langle Im(\boldsymbol{A}(s,:)), Re(\boldsymbol{R}(p,:)), Im(\boldsymbol{A}(o,:))\rangle \\
&+ \langle Re(\boldsymbol{A}(s,:)), Im(\boldsymbol{R}(p,:)), Im(\boldsymbol{A}(o,:))\rangle \\
&- \langle Im(\boldsymbol{A}(s,:)), Im(\boldsymbol{R}(p,:)), Re(\boldsymbol{A}(o,:))\rangle,
\end{aligned} \tag{4.3}$$

with $\boldsymbol{A} \in \mathbb{C}^{|\mathcal{E}|\times\tilde{r}}$ containing the latent representations for all entities and $\boldsymbol{R} \in \mathbb{C}^{|\mathcal{R}|\times\tilde{r}}$ containing the latent representations for all relations. $Re(\cdot)$ and $Im(\cdot)$ denote the real and imaginary part, respectively, and $\bar{\cdot}$ denotes the complex conjugate. By allowing complex numbers for the embedding of the entities, ComplEx

is able to model also non-symmetric relations.

Multi-way NN The multi-way neural network [41, 106] concatenates all embeddings and feeds them to a neural network of the form

$$\boldsymbol{\mathcal{X}}(s,p,o) \approx \boldsymbol{\beta}^T \tanh\left(\boldsymbol{W}\left[\boldsymbol{A}(s,:),\boldsymbol{R}(p,:),\boldsymbol{A}(o,:)\right]+\boldsymbol{b}_1\right)+b_2, \tag{4.4}$$

where $[\cdot,\cdot,\cdot]$ denotes the concatenation of the embedding vectors. The prediction is derived using a Mulilayer Perceptron with the weight matrix $\boldsymbol{W} \in \mathbb{R}^{3\tilde{r}\times z}$, the weight vector $\boldsymbol{\beta} \in \mathbb{R}^z$, and the bias terms $\boldsymbol{b}_1 \in \mathbb{R}^z$, and $b_2 \in \mathbb{R}$.

RESCAL The tensor decomposition RESCAL [107] learns vector embeddings for entities and matrix embeddings for relations. The score function is

$$\boldsymbol{\mathcal{X}}(s,p,o) \approx \boldsymbol{A}(s,:)\cdot\boldsymbol{\mathcal{R}}(:,p,:)\cdot\boldsymbol{A}(o,:)^T, \tag{4.5}$$

with $\cdot$ denoting the dot product, with $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{E}|\times\tilde{r}}$ containing the latent representations for all entities and $\boldsymbol{\mathcal{R}} \in \mathbb{R}^{\tilde{r}\times|\mathcal{R}|\times\tilde{r}}$ containing the latent representations in for of matrices for all relations. RESCAL is a Tucker-2 decomposition with shared factor matrices at the first and third dimension.

Typically, the models are trained using a Bernoulli or a ranking cost function [106]. For our task of visual relationship detection, we will train them slightly differently using a Poisson cost function for modeling count data, as we will show in Section 4.3.2. Another popular link prediction method is TransE [23], however it is not appropriate for modeling count data; thus we are not considering it in this work.

4.2.2 Image Classification and Object Detection

Computer vision methods for image classification and object detection have improved enormously over the last years. Convolutional neural networks (CNN), which apply convolutional filters in a hierarchical manner to an image, have become the standard for image classification. In this work, we use the following two methods.

VGG The VGG-network is a convolutional neural network, which has shown state-of-the-art performance at the Imagenet challenge [133]. It exists in two versions, i.e. the more commonly used VGG-16 with 16 convolutional layers, and VGG-19 with 19 convolutional layers. The model is especially popular due to its uniform architecture. All convolutional layers throughout the network perform a 3×3 convolution, and all pooling layers perform 2×2 max pooling. Each convolution layer is followed by a rectified linear unit (Relu) activation function. The input size to the network is $3 \times 244 \times 244$, which means that the images should have three color channels and a size of 244×244. Thus images are typically scaled and warped to fit this format. In total, the VGG architecture has about 140 million parameters. As networks of this size require many training data and enormous computing resources, models with pre-trained parameters have been made available. The pretrained models are typically trained on the Imagenet dataset which consists of around 1.2 million images and 1000 classes. When the VGG network is applied to a new dataset, only the output layer, which maps to the classes of the specific task, is trained entirely. The last few layers of the pre-trained network are typically fine-tuned using a small learning rate. VGG is also a popular model for deriving image features. These features are taken from the activation of the second last layer, which has a size of 4096.

RCNN The region-based convolutional neural network (RCNN) [59] is a popular object detection method. Given an input image, it proposes bounding boxes, which show visual objects in the image, and their classification. The output is derived in multiple successive steps. First, a selective search algorithm is applied, which proposes around 2000 candidate regions in the image [151]. Selective search gradually combines regions of different scales based on their similarity. The similarity measure takes into account the color, the texture, the size, and the position of the regions. This procedure works in a purely unsupervised manner. The derived regions from the selective search algorithm are then classified into $N + 1$ classes. Therefore, a classifier has to be trained for these particular classes. The classes are the N classes of the dataset plus a background class. By integrating the background class many regions are filtered out. The regions are further filtered using a greedy non-maximum suppression. This method rejects those regions, which

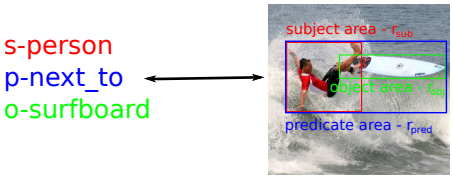


Figure 4.2: The subject and object of the triple relate to two regions in the image, and the predicate relates to the union of the two regions. Image taken from the Stanford Visual Relationship dataset [91].

have a large overlap with a higher scoring region for each class. As a result, a small set of region proposals is derived. A regression model based on CNN features of an image region is finally applied to predict adjustments to the bounding boxes. With these adjustments tighter bounding boxes can be reached. There are two extensions to RCNN, which are mainly faster to compute [58, 119]. However, in our experiments we use the original RCNN, for a fair comparison with [91]. Our focus is on improving visual relationship detection through incorporating semantic modeling rather than on improving computer vision techniques.

4.3 Visual Relationship Detection

In this section, we define the problem of visual relationship detection and describe our proposed models.

4.3.1 General Setting

The training data consists of images and corresponding sets of semantic triples which describe the image. Each semantic triple consists of a subject $s \in \mathcal{E}$, a predicate $p \in \mathcal{R}$, and an object $o \in \mathcal{E}$, where $\mathcal{E}$ is a set of indices which represent visual concepts (e.g. man, horse) and $\mathcal{R}$ is a set of indices which represent relationships between visual concepts (e.g. *riding*, *next_to*). Each triple (s, p, o) is associated with image regions $r_{\text{sub}}, r_{\text{pred}}, r_{\text{obj}}$ which are described by the coordinates of the bounding boxes containing the respective image region. Thereby, r_{sub} describes the region in the image which shows the subject, and r_{obj} describes the region which contains the object. The region r_{pred} is defined as the union of the regions

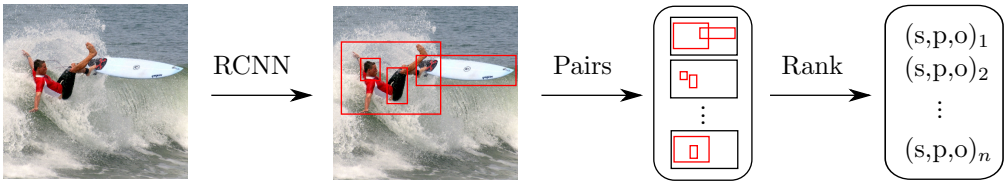


Figure 4.3: The pipeline for deriving a ranked list of triples. Images taken from the Stanford Visual Relationship dataset [91].

r_{sub} and r_{obj}. Each sample of the training data is represented as a six-tuple of the form $(r_{\text{sub}}, r_{\text{pred}}, r_{\text{obj}}, s, p, o)$. Figure 4.2 shows an example of a triple and its corresponding bounding boxes. During training, all triples and their corresponding areas are observed. After model training, the task is to predict the most likely tuples $(r_{\text{sub}}, r_{\text{pred}}, r_{\text{obj}}, s, p, o)$ for a given image.

The methods proposed in this work follow the processing pipeline illustrated in Figure 4.3. Input to the pipeline is a raw image and output a ranked list of triples, each with their corresponding bounding boxes. First, the objects in the image are detected using a RCNN model. The RCNN results in a list of object regions which is of variable length. Then all possible pairs of the detected bounding boxes are built. Finally, for each pair of bounding boxes a corresponding triple is predicted. The predicted triples are ranked according to their confidence score. Finally, the output is a ranked list of triples, which are associated with the regions detected by the RCNN. The procedure for deriving the bounding box pairs is identical to the one used in [91]. In our work, we propose two different models for the last step, which is predicting the triples given the pairs of detected bounding boxes.

4.3.2 Bayesian Fusion Model

The triples are predicted, given the detected region proposals by the RCNN. Our first approach, the Bayesian fusion model, can be divided into a visual model, and semantic model, which are combined in a Bayesian framework. Figure 4.4 shows the basic concept of the approach.

Visual model The visual model consists of two convolutional neural networks, with an VGG-16 architecture [132]. The first CNN which we denote as CNN_e takes

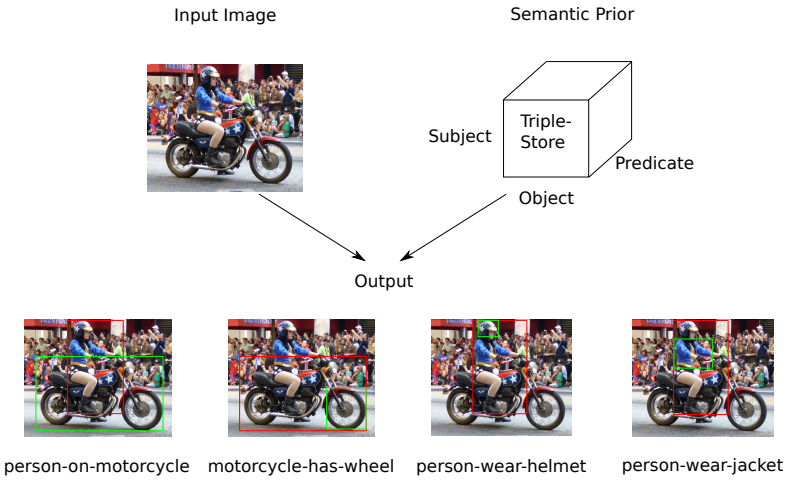


Figure 4.4: In the Bayesian fusion model, the visual component is fused with a semantic prior. Images taken from the Stanford Visual Relationship dataset [91].

as input a subregion of an image, which is defined by a bounding box, and outputs a probability distribution over the visual concepts in $\mathcal{E}$. The second CNN, which we denote as CNN_r takes the union region of two bounding boxes as an input, and outputs a probability distribution over the relationships in $\mathcal{R}$. While training, both CNNs use the regions (bounding boxes) provided in the training data (r_{sub}, r_{pred}, and r_{obj}).

During inference, the CNNs are applied to the regions which are derived from the RCNN. As shown in the processing pipeline in Figure 4.3, the next step is to build all possible pairs out of the regions, which have been detected by the RCNN. In each pair, we denote the first region as r_{sub} and the second as r_{obj}. We apply CNN_e to both regions separately, to derive the classification scores $p(s|r_{\text{sub}}) = CNN_e(r_{\text{sub}})$ and $p(o|r_{\text{obj}}) = CNN_e(r_{\text{obj}})$. Then the union of the regions r_{sub} and r_{obj} which we denote as r_{pred}, is fed to CNN_r to derive the score $p(p|r_{\text{pred}}) = CNN_r(r_{\text{pred}})$.

Probabilistic Semantic Model In the probabilistic semantic model, we train a latent variable model based on tensor decomposition in the following way. In contrast to typical knowledge graph modeling, we do not only have one global graph $\mathcal{G}$, but an instance of a knowledge graph $\mathcal{G}_i$ for every image i. Each triple which appears in a certain image can be described as a tuple (s, p, o, i). The link

prediction model shall reflect the likelihood of a triple to appear in a graph instance, as a prior without seeing the image. By summing over the occurrences in the i-th dimension, we derive the absolute frequency of triples (s, p, o) in the training data, which we denote as $y_{s,p,o}$. We build a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{R}|\times|\mathcal{E}|}$ and set its elements as $\boldsymbol{\mathcal{X}}(s, p, o) = y_{s,p,o}$. We aim to model $\boldsymbol{\mathcal{X}}$ using the link prediction methods described in Section 4.2.1, which approximate the tensor with a decomposition, where $\boldsymbol{\mathcal{X}} \approx \hat{\boldsymbol{\mathcal{X}}}$. As we are dealing with count data, we assume a Poisson distribution on the model output $\hat{\boldsymbol{\mathcal{X}}}$. The log-likelihood for a triple is

$$\begin{aligned}\log p(\boldsymbol{\mathcal{X}}(s,p,o)|(s,p,o),\Theta) = \boldsymbol{\mathcal{X}}(s,p,o)\log\phi(\hat{\boldsymbol{\mathcal{X}}}(s,p,o)) \\ -\phi(\hat{\boldsymbol{\mathcal{X}}}(s,p,o)) - \log(\boldsymbol{\mathcal{X}}(s,p,o)!),\end{aligned} \tag{4.6}$$

where Θ are the model parameters of the link prediction method and ϕ is the activation function for the Poisson distribution, namely

$$\phi(\hat{\boldsymbol{\mathcal{X}}}(s,p,o)) = \exp(\hat{\boldsymbol{\mathcal{X}}}(s,p,o)). \tag{4.7}$$

We train the model by minimizing the negative log-likelihood. In the objective function the last term $\log(\boldsymbol{\mathcal{X}}(s, p, o)!)$ can be neglected, as it does not depend on the model parameters. Thus, the cost function for the whole training dataset becomes

$$cost = \sum_{(s,p,o)} \phi(\hat{\boldsymbol{\mathcal{X}}}(s,p,o)) - \boldsymbol{\mathcal{X}}(s,p,o)\log\phi(\hat{\boldsymbol{\mathcal{X}}}(s,p,o)). \tag{4.8}$$

Using this framework, we can train any of the link prediction methods described in Section 4.2.1, by plugging the prediction into the cost function and minimizing the cost function using a gradient-descent based optimization algorithm. In this work, we use Adam, a recently proposed first-order gradient-based optimization method with adaptive learning rate [81].

Probabilistic Joint Model In the last step of the pipeline in Figure 4.3, which we denote as ranking step, we need to combine the scores from the visual model with the scores from the semantic model. For joining both, we propose a probabilistic model for the interaction between the visual and the semantic part. Figure 4.5 visualizes the joint model for all variables in a probabilistic graphical model.

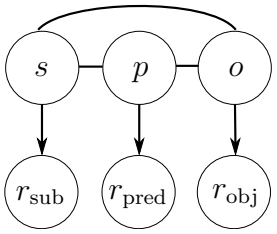


Figure 4.5: The probabilistic graphical model describes the interaction between the visual and the semantic part in the Bayesian fusion model. We assume the image regions r_{sub}, r_{pred} and r_{obj} to be given by the RCNN, and infer the underlying s, p, o triples.

The joint distribution factors as

$$p(s, p, o, r_{\text{sub}}, r_{\text{pred}}, r_{\text{obj}}) = p(s, p, o) \cdot p(r_{\text{sub}}|s) \cdot p(r_{\text{pred}}|p) \cdot p(r_{\text{obj}}|o). \tag{4.9}$$

We can divide the joint probability of Equation 4.9 into two parts. The first part is $p(s, p, o)$, which models semantic triples. The second part is $p(r_{\text{sub}}|s) \cdot p(r_{\text{pred}}|p) \cdot p(r_{\text{obj}}|o)$, which models the visual part given the semantics.

Following [144, 146], we derive the joint probability of the triples $p(s, p, o)$ using a Boltzmann distribution. With the energy function $E(s, p, o) = -\log \phi(\hat{\boldsymbol{\mathcal{X}}}(s, p, o))$ the probability for the triples becomes

$$p(s, p, o) = \frac{\phi(\hat{\boldsymbol{\mathcal{X}}}(s, p, o))}{\sum_{s' \in \mathcal{E}} \sum_{p' \in \mathcal{R}} \sum_{o' \in \mathcal{E}} \phi(\hat{\boldsymbol{\mathcal{X}}}(s', p', o'))}. \tag{4.10}$$

The visual models described in the previous section, model the probabilities $p(s|r_{\text{sub}})$, $p(p|r_{\text{pred}})$, and $p(o|r_{\text{obj}})$. By applying Bayes rule to Equation 4.9 and conditioning on the image regions we get

$$p(s, p, o|r_{\text{sub}}, r_{\text{pred}}, r_{\text{obj}}) \propto p(s, p, o) \cdot \frac{p(s|r_{\text{sub}}) \cdot p(p|r_{\text{pred}}) \cdot p(o|r_{\text{obj}})}{p(s) \cdot p(p) \cdot p(o)}. \tag{4.11}$$

We derive the additional terms of the denominator $p(s)$, $p(p)$, $p(o)$ through

marginalization of $p(s, p, o)$ and Laplace smoothing, such that

$$p(s) = \frac{\sum_{p \in \mathcal{R}} \sum_{o \in \mathcal{E}} p(s, p, o) + \alpha}{\alpha + 1}, \tag{4.12}$$

$$p(p) = \frac{\sum_{s \in \mathcal{E}} \sum_{o \in \mathcal{E}} p(s, p, o) + \alpha}{\alpha + 1}, \tag{4.13}$$

$$p(o) = \frac{\sum_{s \in \mathcal{E}} \sum_{p \in \mathcal{R}} p(s, p, o) + \alpha}{\alpha + 1}, \tag{4.14}$$

with α being a hyperparameter. For each image, we derive the region candidates r_{sub}, r_{pred}, r_{obj} from the RCNN. The final prediction score for each triple given the bounding boxes is

$$p(s, p, o | r_{\text{sub}}, r_{\text{pred}}, r_{\text{obj}}) \propto \phi(\hat{\boldsymbol{\mathcal{X}}}(s, p, o)) \frac{CNN_e(r_{\text{sub}}) \cdot CNN_r(r_{\text{pred}}) \cdot CNN_e(r_{\text{obj}})}{p(s) \cdot p(p) \cdot p(o)}. \tag{4.15}$$

For each pair of bounding boxes, we pick the triple with the highest probability as a final prediction.

4.3.3 Conditional Multi-way Model

We now propose an alternative model for the task of visual relationship detection, which does not include an explicitly trained prior. It rather learns the semantic prior implicitly in its latent representations during training. This model builds upon the same visual pipeline as the Bayesian Fusion model, but uses another model to derive the final ranking of image related triples.

We again assume the candidate boxes being detected by the RCNN, and predict semantic triples for all possible pairs of bounding boxes. For a bounding box pair $(r_{\text{sub}}, r_{\text{obj}})$, we derive the subject s and the object o by applying the CNN classifier CNN_e to the respective regions, which provides us with the probabilities $p(s|r_{\text{sub}}) = CNN_e(r_{\text{sub}})$ and $p(o|r_{\text{obj}}) = CNN_e(r_{\text{obj}})$.

To derive a triple from the pair of classified bounding boxes, we need to infer the relationship between the two visual objects. We propose a multi-way model conditioned on a specific s, o, and r_{pred}, which is again the union of the regions

(r_{sub} and r_{obj}), to derive the corresponding p. As p is sampled from a categorical random variable, we model the probability distribution over all possible p using the softmax function, such that

$$p(p|s, o, r_{\text{pred}}) = \frac{\exp(f_p(s, o, r_{\text{pred}}))}{\sum_{\tilde{p} \neq p} \exp(f_{\tilde{p}}(s, o, r_{\text{pred}}))}. \tag{4.16}$$

We implement the function $f_p(s, o, r_{\text{pred}})$ with a multi-way neural network model as introduced in Section 1.2.3 and Equation 1.22. The multi-way neural network learns latent representations for the input entities such that,

$$f_p(s, o, r_{\text{pred}}) = \boldsymbol{w}_p^T \tanh(\boldsymbol{W} \left[\boldsymbol{a}_s, \boldsymbol{a}_o, \boldsymbol{e}_{r_{\text{pred}}}\right] + \boldsymbol{b}) + b_p, \tag{4.17}$$

with $\boldsymbol{a}_s, \boldsymbol{a}_o \in \mathbb{R}^d$ being latent vector representations for the visual concepts, $\boldsymbol{e}_{r_{\text{pred}}} \in \mathbb{R}^d$ being a latent representation vector for the image patch r_{pred}, and $[\cdot, \cdot, \cdot]$ denoting the concatenation operation. For $\boldsymbol{a}_s$ and $\boldsymbol{a}_o$ the representations are learned directly and stored in a lookup table. For deriving a representation of the predicate region r_{pred}, we model $\boldsymbol{e}_{r_{\text{pred}}} = \boldsymbol{M}\boldsymbol{h}_{r_{\text{pred}}}$, where $\boldsymbol{h}_{r_{\text{pred}}}$ is the activation of the second last layer of the VGG network CNN_r with the image region r_{pred} as input. The matrix M maps the latent representation of the VGG network to the rank of the multi-way model. The probabilities for p are derived by applying a multilayer perceptron with the additional parameters $\boldsymbol{W} \in \mathbb{R}^{3d \times z}, \boldsymbol{w}_p \in \mathbb{R}^z, \boldsymbol{b} \in \mathbb{R}^z, b_p \in \mathbb{R}$. This is a forward model which can be trained end-to-end.

To derive a single prediction for each pair of bounding boxs, we pick the subject $\hat{s} = \arg\max p(s|r_{\text{sub}})$, object $\hat{o} = \arg\max p(o|r_{\text{sub}})$ and predicate $\hat{p} = \arg\max p(p|r_{\text{pred}}, \hat{s}, \hat{o})$ with the highest probabilities. After deriving $\hat{s}$, $\hat{p}$, and $\hat{o}$ we derive the final confidence score for the triple $(\hat{s}, \hat{p}, \hat{o})$ given the input regions $(r_{\text{sub}}, r_{\text{pred}}, r_{\text{obj}})$ as

$$p(\hat{s}, \hat{p}, \hat{o}|r_{\text{sub}}, r_{\text{pred}}, r_{\text{obj}}) = p(\hat{s}|r_{\text{sub}}) \cdot p(\hat{o}|r_{\text{obj}}) \cdot p(\hat{p}|r_{\text{pred}}, \hat{s}, \hat{o}). \tag{4.18}$$

Figure 4.6 shows the model schematically. The visual part from the raw image to the pairs of bounding boxes is identical to the Bayesian fusion model. The next step shows the forward processing of the conditional multi-way model, which

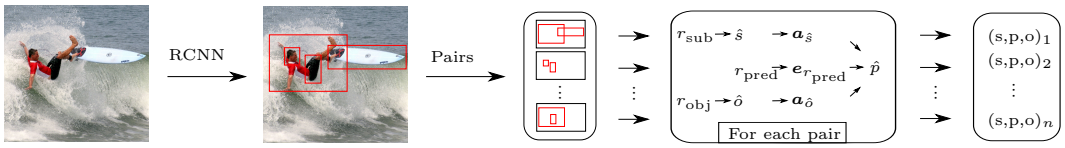


Figure 4.6: The processing pipeline for the conditional multi-way model. The difference to the Bayesian fusion model is in how the triples are derived given the region pairs. Images taken from the Stanford Visual Relationship dataset [91].

derives first $\hat{s}$ and $\hat{o}$ and then the predicate $\hat{p}$. Finally, one triple is derived for each pair of bounding boxes and the triples are ranked according to their confidence score.

4.4 Related Work

Lu et al. [91] The Stanford computer vision group where the first proposing a model to large scale visual relationship detection. The model was applied to a dataset containing 700000 possible triples. The approach of Lu et al. can be seen as a multi-modal approach as they use the information provided in the annotated image dataset, plus word embeddings which are derived from a word2vec model trained on an external text corpus. The model consists of a visual module and a language module. The visual module uses an RCNN for object detection to derive candidate regions. Further, a VGG-16 is applied to the detected regions for obtaining object classification scores for each region. A second VGG, which classifies relationships, is applied to the union of pairs of regions. This vision module is identical to the approach we use in our models. The difference of our models to the model of Lu et al. is in how the triple ranking is derived. Our conditional multi-way model has some similarities with this approach, since it also predicts the predicate of a triple given the representations of the subject and the object. One difference is, that in our conditional multi-way model we also integrate the latent representation of the image region of the predicate. The other main differences are that in our model the embeddings for subject and object are learned entirely during training, without requiring any external data sources. Further, we implemented the multi-way model as a multilayer perceptron, where as

in Lu et al. it is implemented using a linear model. In [91] the relationship between two entities are predicted given the word embeddings of the detected subject and object. Lu et al. add a regularization term, which enforces to give similar scores to similar triples. The similarity between triples is measured by the sum of the cosine distances between the word2vec embeddings of the visual objects and the relations of two triples.

Visual TransE [23] More recently, Zhang et al. [23] have published an approach which combines a visual model with the translational embedding model [23] for the task of visual relationship detection. This approach also builds upon an object detection method, which derives region proposals for objects. Based on the image regions, a feature extraction method is proposed which extracts the class of the object in the region, the location of the region, and a visual feature vector derived from a CNN. The learned visual features are then projected to a relation space, where the translational embedding model is applied to predict the most likely relation between two visual objects.

Earlier Work Some earlier work on visual relationship detection was concerned with learning spatial relationships between objects, however with a very limited set of relations consisting of four spatial directions [53, 63]. Other related work attempted to learn actions and object interactions of humans in videos and images [122, 168, 94, 64, 167, 116]. Full visual relationship detection has been demonstrated in [126, 34, 30], however, also with only small amounts of possible triples. In [30], an ontology over the visual concepts is defined and combined with a neural network approach to maintain semantic consistency. In the Google Knowledge Vault project [41], semantic triples have been extracted from raw text. The approach fuses the confidence scores from the text-based extraction methods with the scores derived from a factorization model of existing knowledge bases. Incorporating the probabilistic semantic prior significantly improved the retrieved results.

Related Tasks Visual relationship detection is also related to visual caption generation, which recently gained considerable popularity among the deep learning

community. In visual caption generation, an image caption consisting of natural text, is generated given an input image, see e.g. [164]. The main difference to visual relationship detection is, that the output in visual relationship detection is more structured (a set of triples versus natural text) and thus it is more appropriate for further automatic processing, e.g. semantic querying, whereas natural text is understandable by humans, but difficult to process by machines. Another important task, which is related to visual relationship detection, is visual question answering, see e.g. [159, 54, 163]. The main difference is that only the information necessary to answer a specific query, e.g. "What is the color of the shirt?" is extracted from the image. In visual relationship detection, all semantic information for a scene is derived, and the querying is applied in a post-processing step.

Cognitive Interpretation In [144, 146, 145], the connection of tensor decomposition models to the perception and memory system in the human brain has been discussed. The human brain needs to interpret information, which is continuously acquired by the visual system. Therefore, it contains internal representations of entities, which are used by the perceptual and the main declarative memory systems [150]. Our proposed Bayesian fusion model can more directly be related to the Bayesian brain hypothesis, as being pursued by many research teams, whereas the conditional multi-way model is more closely related to the tensor memory hypothesis [145]. A further discussion on the relationship can be found in [11].

4.5 Experiments

We evaluate our proposed methods on the recently published Stanford Visual Relationship dataset [91], and compare against the state-of-the-art methods in visual relationship detection. As in [91] we will divide the setting into two parts: First an evaluation on how well the methods perform when predicting all possible triples and second only evaluating on triples, which did not occur in the training data. This setting is also referred to as zero-shot learning, as the model has not seen any training images containing the triples which are used for evaluation.

4.5.1 Dataset

The dataset consists of 5000 images. The semantics are described by triples, consisting of 100 entity types, such as *motorcycle, person, surfboard, watch*, etc. and 70 relations, e.g. *next_to, taller_than, wear, on*, etc. The entities correspond to visual objects in the image. For all *subject* and *object* entities the corresponding regions in the image are given. Each image contains in average 7.5 triples, which describe the scene. In total, there are 37993 triples in the dataset. The dataset is split into 4000 training and 1000 test images. We use the same data split as in [91] and [172], thus we can directly compare our results. There are 1877 triples, which only occur in images from the test set but not in the training set. We will use these triples for the zero-shot analysis.

4.5.2 Methods

We compare both, the Bayesian fusion model and the conditional multi-way model, against the state-of-the-art methods from [91] and [172] in the task of predicting semantic triples from images. As we use the same evaluation setting as in [91] and [172] we directly compare our achieved results with the results reported in these papers. We compare against the purely visual model in [91] which we denote as *Lu et al. V* and the full model including the language prior and the regularization term. We denote this method as *Lu et al. full*. Furthermore, we denote the visual translational embedding model from [172] as *VTransE*. For the Bayesian fusion model we apply the four different factorization methods presented in Section 4.2.1.

4.5.3 Setting

In our experiments, we consider the same three evaluation settings as in [91], which are as follows.

Phrase detection In phrase detection the task is to give a ranking of likely triples plus the corresponding regions for the *subject* and *object* of the triple. The bounding boxes are derived from the RCNN. A triple with its corresponding bounding boxes is considered correctly detected, if the triple is identical to the

ground truth, and if the union of the bounding boxes has at least 50 percent overlap with the union of the ground truth bounding boxes.

Relationship detection The second setting, which is also considered in [91] is relationship detection. It is similar to phrase detection, but with the difference that it is not enough when the union of the bounding boxes is overlapping by at least 50 percent. Instead, both the bounding box of the *subject* and the bounding box of the *object* need at least 50 percent of overlap with their ground truth.

Predicate detection In predicate detection, it is assumed that *subject* and *object* are given, and only the correct *predicate* between both needs to be predicted. Therefore, we use the ground truth bounding boxes with the respective labels for the objects instead of the bounding boxes derived by the RCNN. This separates the problem of object detection from the problem of predicting the correct relationships between the visual objects.

For each test image, we create as many triples as there are pairwise combinations of detected bounding boxes. For each pair of bounding boxes exactly one triple is predicted. These triples are ranked according to their confidence score. Similar to [91] we report the recall at the top 100 elements of the ranked list and the recall at top 50. Note, that there are 700000 possible triples, out of which the correct triples need to be ranked on top. When training the semantic model, we hold out 5 percent of the nonzero triples as a validation set. We determine the optimal rank for the link prediction methods based on that hold-out set. For the visual model (RCNN and VGG) we use the predicted regions and classification scores as provided by [91]. In order to get more stable results in the tensor decompositions, we train them 20 times and average over the reconstructed tensor.

We also include an experimental setting, where we only evaluate on triples, which had not been observed in the training data. This setting reveals the generalization ability of the semantic model. The test set contains 1877 of these triples. We evaluate based on the same settings as in the previous section, however for the recall we only count how many of the unseen triples are retrieved.

Task	Phrase Det.		Rel. Det.		Predicate Det.	
Evaluation	R@100	R@50	R@100	R@50	R@100	R@50
Lu et al. V [91]	2.61	2.24	1.85	1.58	7.11	7.11
Lu et al. full [91]	17.03	16.17	14.70	13.86	47.87	47.87
VTransE [172]	22.42	19.42	15.20	14.07	-	-
Cond. Multi-way Model	17.71	15.79	15.37	13.72	47.93	47.62
Bayes. Fusion - RESCAL	19.17	18.16	16.88	15.88	52.71	52.71
Bayes. Fusion - Multi-wayNN	18.88	17.75	16.65	15.57	51.82	51.82
Bayes. Fusion - ComplEx	19.36	18.25	17.12	16.03	53.14	53.14
Bayes. Fusion - DistMult	15.42	14.27	13.64	12.54	42.18	42.18

Table 4.1: Results for visual relationship detection. We report Recall at 50 and 100 for four different validation settings.

4.5.4 Results

Table 4.1 shows the results for visual relationship detection. In the first three lines we report the results from [91] and [172]. The first row shows the results, when only the visual part of the model is applied. This model performs poorly, in all three settings. The full model in the second row adds the language prior to it and also some regularization terms during training, which are described in more detail in [91]. This drastically improves the results. As expected the recall at top 100 is better than at top 50, however the difference is rather small, which shows that most of the correctly predicted triples are ranked quite high. The results for predicate detection are much better than for the other settings. This shows that one of the main problems in visual relationship detection is the correct detection and classification of the visual objects. The Visual Tanslation Embedding model (Visual TransE) outperforms the model from Lu et al. in the setting of phrase detection and relationship detection. The predicate detection setting has not been reported by [172]. Our conditional multi-way model outperforms the language prior model in some settings and achieves very similar results in the others. In relationship detection the performance is comparable to the Visual TransE model,

Task	Phrase Det.		Rel. Det.		Predicate Det.	
Evaluation	R@100	R@50	R@100	R@50	R@100	R@50
Lu et al. V [91]	1.12	0.95	0.78	0.67	3.52	3.52
Lu et al. full [91]	3.75	3.36	3.52	3.13	8.45	8.45
VTransE [172]	3.51	2.65	2.14	1.71	-	-
Cond. Multi-way Model	5.73	5.39	5.22	4.96	14.32	14.32
Bayes. Fusion RESCAL	6.59	5.82	6.07	5.30	16.34	16.34
Bayes. Fusion Multi-wayNN	6.93	5.73	6.24	5.22	16.60	16.60
Bayes. Fusion ComplEx	6.50	5.73	5.82	5.05	15.74	15.74
Bayes. Fusion DistMult	4.19	3.34	3.85	3.08	12.40	12.40

Table 4.2: Results for the zero shot learning experiments. We report Recall at 50 and 100 for four different validation settings.

in the phrase detection setting Visual TransE clearly outperforms the conditional multi-way model. In the last four rows, we report the results of the Bayesian fusion model. We compare the results for the integration of the four link prediction methods described in Section 4.2.1. We see that with all link prediction methods the model performs constantly better than the method proposed by [91], except for DistMult. The poor performance of DistMult might result from the fact, that it assumes symmetric scores when subject and object are exchanged. For relationship detection, which is the most challenging setting, ComplEx achieves the best results, with a recall of 17.12 and 16.03 for the top 100 and top 50 results respectively. RESCAL performs slightly better than the multi-way neural network in all evaluation settings. The best performing Bayesian fusion model clearly outperforms the VTransE model on the relationship detection task. In phrase detection, however, VTransE performs best with a result of 22.42 and 19.42 for the recall at 100 and 50, respectively.

The optimal rank for the conditional multi-way model has been found at 20 on the validation set. For ComplEx, RESCAL, and multi-way neural network a rank of of 12 has been found to be best. For the DistMult model a rank of 20

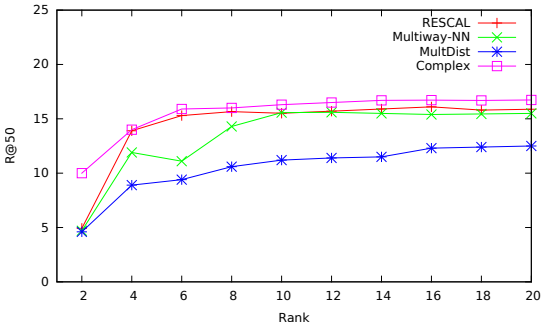


Figure 4.7: Recall at 50 as a function of the rank.

was found to be optimal. Figure 4.7 shows the recall at 50 on the test set for our different variants as a function of the rank. We see that the performances of ComplEx and RESCAL converge relatively quickly to a recall of around 16. The multi-way neural network converges a bit slower, to a slightly smaller maximum. DistMult converges slower and to a much smaller maximum recall of 12.5.

Table 4.2 shows the results for the zero-shot experiments. This task is much more difficult, which can be seen by the huge drop in recall. However, also in this experiment, including the semantic model significantly improves the prediction. For the first three settings, the best performing method, which is the multi-way neural network, almost retrieves twice as many correct triples, as the state-of-the-art model of [91]. Especially, for the predicate detection, which assumes the objects and subjects to be given, a relatively high recall of 16.60 can be reached. In the zero-shot setting for predicate detection even the integration of the worst performing semantic model DistMult shows significantly better performance than the state-of-the-art method. These results clearly show that our model is able to infer also new likely triples, which have not been observed in the training data. This is one of the big benefits of the link prediction methods. VTransE performes poorly in the zero-shot setting. This is astonishing, as the integration of the Tranlational Embedding (TransE) model should have a similar effect as the tensor decomposition models, proposed in this paper. A reason why this is not the case could be, is that the TransE model is to simple to capture the existing semantics in the dataset. The language prior allows the model to generalize to unseen triples. However, our experiments show that integrating state-of-the-art

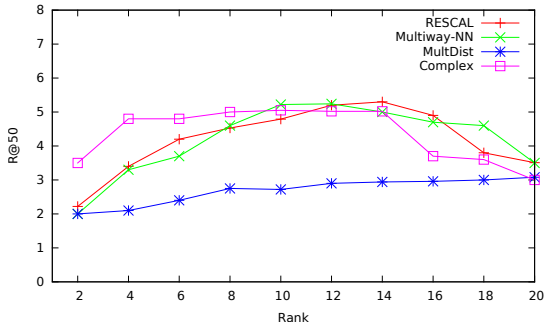


Figure 4.8: Recall at 50 as a function of the rank for the zero-shot setting.

link prediction methods for modeling semantics based on tensor decomposition is more appropriate for improving general prediction and generalization to unseen triples.

For the zero-shot settings the same parameters for the rank have been used, as found optimal on the validation set for the first setting. To illustrate the importance of the rank in the zero-shot setting, Figure 4.8 shows the recall at 50 on the zero-shot test set as a function of the rank. As expected, the models start to overfit in the zero-shot setting if the rank is to high. With a limited rank the models have less freedom for explaining the variation in the data; this forces them to focus more on the underlying structure, which improves the generalization property. ComplEx, which has more parameters due to the complex valued embeddings, performs best with small ranks and reaches the maximum at a rank of around 8. multi-way neural network reaches the maximum at a rank of 10 and RESCAL at a rank of 14. The highest recall is achieved by RESCAL at 5.3.

4.6 Conclusion

In this chapter, we presented two novel approaches for including semantic knowledge into visual relationship detection. Both approaches combine standard computer vision methods for perceptual modeling, with latent variable models for semantic modeling. We combine a state-of-the-art computer vision procedure with latent variable models for link prediction, in order to enhance the modeling of relationships among visual objects. In the first approach, we proposed a probabilistic

framework, in form of a Bayesian fusion model, for integrating both the semantic prior and the computer vision algorithms into a joint model. By including a statistical semantic model, the predictive quality can be enhanced significantly. Especially the prediction of triples, which have not been observed in the training data, can be enhanced through the generalization properties of the semantic link prediction methods. The recall of the best performing link-prediction method in the zero-shot setting is almost twice as high as the state-of-the art method. The second approach uses a conditional multi-way model, which is inspired by link prediction methods. For the prediction of triples, which have not been observed in the training data, the performance of the second approach is on par with the first approach, as its structure helps to generalize to unobserved triples, without including a separately trained prior for the semantic triples. The semantic prior is implicitly represented in the learned latent representations of the involved entities. Both approaches form statistical models on the class level, and can thus generalize to new images. This is in contrast to typical knowledge graph models, where nodes correspond to specific instances. Our experiments show, that the interaction of semantic and perceptual models can support each other to derive better predictive accuracies. The improvement over the state-of-the-art vision model shows that visual relationship detection can not only be improved by better computer vision methods, but also by multi-modal approaches, in particular, the integration of a component, which models the semantic structures. The developed methods show great potential also for broader application areas, where both semantic and sensory data needs to be fused.

Chapter 5

Conclusion

In this thesis, we examined various approaches to information fusion in supervised machine learning. In this chapter, we want to summarize the main aspects and discuss interesting directions for future work and application areas.

5.1 Summary and Discussion

We considered three different aspects of information fusion in supervised machine learning and demonstrated the effectiveness of the elaborated models on different applications, such as modeling distributed sensor networks, feed-forward robot control, and visual relationship detection in images.

In contrast to traditional model agnostic fusion approaches, which either take an early or a late fusion approach, the concept of representation learning allows for performing the information fusion at an intermediate level, i.e., the level of latent representations derived from the raw input data. The proposed models in this thesis are all trained in a supervised manner. In this way, the representations are implicitly optimized so as to be the most advantageous for the predictive task. The representations capture the relevant factors for predicting the output as a side effect while modeling a conditional predictive model. This is in contrast to unsupervised representation learning, where the representations are trained to capture the latent explanatory factors of the observed input data itself.

In Chapter 2, a single representation was learned for each input channel, which

was then used as a predictor for multiple decoders. In this way, the model learns to extract not only the necessary information for predicting a single time series but also information that is a potential predictor for related data streams. The complete model (including the encoders, the interconnection layer with an attention mechanism, and the decoders) is trained in an end-to-end fashion so that it learns to predict a sequence of future behavior. In this way, the encoders are not trained to be a good model of the input sequence but rather to produce a representation that leads to good output sequence predictions. An important property of RNNs, which we use as encoder models, with regard to data stream modeling is their ability to selectively encode information from previous time steps in a compact hidden state representation that is continuously updated at every time step. Thus, RNNs are good candidates for the efficient processing of streaming data. Using an attention mechanism, which has previously shown to be advantageous in various tasks of natural language processing, for the fusion of the latent representations allows for adjusting the fusion processes dynamically to the current system state. Nevertheless, the model can only learn to extract and effectively process signals that have occurred in the training data.

In Chapter 3, we examined how to efficiently model the interactions in multi-way data using tensor decompositions, which allow for the learning of latent representations for each input dimension. By fusing the learned latent representations in an efficient way, predictions for each of the exponentially many combinations of input signals can be derived. A similar approach was taken in the models for visual relationship detection described in Chapter 4. In the Bayesian fusion approach, the semantic model is trained on the marginalized count data of semantic triples, derived from a visual relation extraction dataset. In this way, the learned representations capture the semantic world as described by the triples as they appear in the dataset. In the second approach (the conditional multi-way model), the representations for the various visual concepts are directly learned in an end-to-end model, which is trained to predict the triples given the images. One of the main advantages of both models is their ability to generalize to triples that have never been observed in the training data, but which seem reasonable in the realm of the dataset. The models, however, are not able to generalize to different semantic worlds, which are driven by distinct semantic rules.

5.2 Future Directions and Applications

Information fusion is an important aspect of machine learning research. Since representation learning has demonstrated excellent performance in a variety of machine learning tasks, there is a clear trend towards model-based fusion approaches. The concept of performing information fusion on the basis of latent representations is a promising framework for modeling complex systems. Such an approach shows promising results, but it also comes with some problems that need to be addressed in future work.

Most existing artificial intelligence (AI) applications are specific to certain cognitive tasks. However, the human brain, sometimes referred to as the only real existing intelligent system, is very good at combining information from various sources. Therefore, the fusion aspect is also important for AI systems. Our proposed approaches to visual relationship detection detailed in Chapter 4, which integrate semantic reasoning and visual perception into a single model, are one step towards this goal. A limiting aspect for training models on such complex tasks is often the availability of training data. In this regard, most successful machine learning applications nowadays are trained using supervised learning and, in order to train the models, large amounts of labeled data needs to be available, which is often expensive to gather. The datasets that are available in the research community often focus on very specific problems and thus do not always reflect the complexity of real-world challenges. Therefore, for more complex settings, some effort needs to go into building high-quality labeled data sets. For example, for visual relationship detection, as discussed in Chapter 4, a larger data set would be advantageous to boost the performance of the models. In many real-world scenarios, however, large and extensively labeled data is not always available. Thus, an important direction for future machine learning research is to acquire universal models that can be transferred between domains. This area of research, referred to as transfer learning, is therefore crucial for putting more deep-learning applications into practice.

With the dissemination of machine learning models in many industries, information fusion is also becoming increasingly important in technical applications. In this sense, new application areas for information fusion arise in IoT applica-

tions and sensor networks. Even with today's advanced computation facilities, fast models are still desirable, especially in embedded real-time systems. Our proposed information fusion architecture in Chapter 2 offers an example of how a model-based distributed architecture could look. Models such as the fast multi-linear ones proposed in Chapter 3 may also find applications in areas with limited compute resources, as shown in the example application of the control of a robot arm.

Another problem that is particularly relevant to modern deep-learning architectures is their missing interpretability, which makes it difficult to apply the models in safety-critical applications. The fast multi-linear models proposed in Chapter 3 take a step in this direction by creating an efficient way of computing conditional odds-ratios, which are used to explain the model output. The neural attention mechanisms used in Chapter 2 are another example of obtaining more interpretability in neural network models. However, if the learned representations are highly entangled, human interpretation becomes difficult. Therefore, the learning of disentangled representations is highly desirable, where a single neuron, or at least a subgroup of neurons, exclusively represents a certain semantic aspect of the modeled system.

Bibliography

[1] Evrim Acar, Canan Aykut-Bingol, Haluk Bingol, Rasmus Bro, and Bülent Yener. Multiway analysis of epilepsy tensors. *Bioinformatics*, 23(13):i10–i18, 2007.

[2] Pradeep K Atrey, M Anwar Hossain, Abdulmotaleb El Saddik, and Mohan S Kankanhalli. Multimodal fusion for multimedia analysis: a survey. *Multimedia systems*, 16(6):345–379, 2010.

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

[4] Brett W Bader, Michael W Berry, and Murray Browne. Discussion tracking in enron email using parafac. In *Survey of Text Mining II*, pages 147–163. Springer, 2008.

[5] Andrew D Bagdanov, Marco Bertini, Alberto Del Bimbo, Giuseppe Serra, and Carlo Torniai. Semantic annotation and retrieval of video events using multimedia ontologies. In *Semantic Computing, 2007. ICSC 2007. International Conference on*, pages 713–720. IEEE, 2007.

[6] Mohammad Taha Bahadori, Qi Rose Yu, and Yan Liu. Fast multivariate spatio-temporal analysis via low rank tensor learning. In *Advances in neural information processing systems*, pages 3491–3499, 2014.

[7] Mohammad Taha Bahadori, Qi Rose Yu, and Yan Liu. Fast multivariate spatio-temporal analysis via low rank tensor learning. In *Advances in neural information processing systems*, pages 3491–3499, 2014.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[9] Stephan Baier, Denis Krompass, and Volker Tresp. Learning representations for discrete sensor networks using tensor decompositions. *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2016.

[10] Stephan Baier, Yunpu Ma, and Volker Tresp. Improving visual relationship detection using semantic modeling of scene descriptions. *International Semantic Web Conference*, 2017.

[11] Stephan Baier, Yunpu Ma, and Volker Tresp. Improving information extraction from images with learned semantic models. *International Joint Conference on Artificial Intelligence*, 2018.

[12] Stephan Baier, Sigurd Spieckermann, and Volker Tresp. Attention-based information fusion using multi-encoder-decoder recurrent neural networks. *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2017.

[13] Stephan Baier, Sigurd Spieckermann, and Volker Tresp. Tensor decompositions for modeling inverse dynamics. *Proceedings of the Congress of the International Federation of Automatic Control*, 2017.

[14] Stephan Baier and Volker Tresp. Factorizing sparse tensors for supervised machine learning. *NIPS workshop on tensor methods*, 2016.

[15] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. Multimodal machine learning: A survey and taxonomy. *arXiv preprint arXiv:1705.09406*, 2017.

[16] Hichem Bannour and Céline Hudelot. Towards ontologies for image interpretation and annotation. In *Content-Based Multimedia Indexing (CBMI), 2011 9th International Workshop on*, pages 211–216. IEEE, 2011.

[17] Azzedine Bendjebbour, Yves Delignon, Laurent Fouque, Vincent Samson, and Wojciech Pieczynski. Multisensor image segmentation using dempster-shafer fusion in markov fields context. *IEEE Transactions on Geoscience and Remote Sensing*, 39(8):1789–1798, 2001.

[18] 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.

[19] Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41, 2007.

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

[21] Stephan Bloehdorn, Kosmas Petridis, Carsten Saathoff, Nikos Simou, Vassilis Tzouvaras, Yannis Avrithis, Siegfried Handschuh, Yiannis Kompatsiaris, Steffen Staab, and Michael G Strintzis. Semantic annotation of images and videos for multimedia analysis. In *European Semantic Web Conference*, pages 592–607. Springer, 2005.

[22] 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.

[23] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

[24] Henrik Boström, Sten F Andler, Marcus Brohede, Ronnie Johansson, Alexander Karlsson, Joeri Van Laere, Lars Niklasson, Marie Nilsson, Anne Persson, and Tom Ziemke. On the definition of information fusion as a field of research, 2007.

[25] George EP Box, Gwilym M Jenkins, and Gregory C Reinsel. *Time series analysis: forecasting and control*, volume 734. John Wiley & Sons, 2011.

[26] David S Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, DTIC Document, 1988.

[27] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, et al. Api design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.

[28] Etienne Burdet and Alain Codourey. Evaluation of parametric and nonparametric nonlinear adaptive controllers. *Robotica*, 16(01):59–73, 1998.

[29] J Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 35(3):283–319, 1970.

[30] Na Chen, Qian-Yi Zhou, and Viktor Prasanna. Understanding web images by object relation network. In *Proceedings of the 21st international conference on World Wide Web*, pages 291–300. ACM, 2012.

[31] Wei-Chen Cheng, Stanley Kok, Hoai Vu Pham, Hai Leong Chieu, and Kian Ming A Chai. Language modeling with sum-product networks. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[32] KyungHyun Cho, Aaron C. Courville, and Yoshua Bengio. Describing multimedia content using attention-based encoder-decoder networks. *arXiv 1507.01053*, 2015.

[33] 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. In *Proceedings of the Empirical Methods in Natural Language Processing*, 2014.

[34] Wongun Choi, Yu-Wei Chao, Caroline Pantofaru, and Silvio Savarese. Understanding indoor scenes using 3d geometric phrases. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 33–40, 2013.

[35] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

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

[37] Jerome Connor, Les E Atlas, and Douglas R Martin. Recurrent networks and narma modeling. In *Advances in Neural Information Processing Systems*, pages 301–308, 1992.

[38] John J Craig. *Introduction to robotics: mechanics and control*, volume 3. Pearson Prentice Hall Upper Saddle River, 2005.

[39] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[40] Dua Dheeru and Efi Karra Taniskidou. Uci machine learning repository, 2017.

[41] 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.

[42] Kenji Doya. *Bayesian brain: Probabilistic approaches to neural coding*. MIT press, 2007.

[43] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[44] Fabon Dzogang, Marie-Jeanne Lesot, Maria Rifqi, and Bernadette Bouchon-Meunier. Early fusion of low level features for emotion mining. *Biomedical informatics insights*, 5:BII–S8973, 2012.

[45] Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin Riedmiller, and Wolfram Burgard. Multimodal deep learning for robust rgb-d object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 681–687. IEEE, 2015.

[46] Marc O Ernst and Heinrich H Bülthoff. Merging the senses into a robust percept. *Trends in cognitive sciences*, 8(4):162–169, 2004.

[47] 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.

[48] 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.

[49] 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.

[50] Cristóbal Esteban, Volker Tresp, Yinchong Yang, Stephan Baier, and Denis Krompaß. Predicting the co-evolution of event and knowledge graphs. In *Information Fusion (FUSION), 2016 19th International Conference on*, pages 98–105. IEEE, 2016.

[51] Georgios Evangelopoulos, Athanasia Zlatintsi, Alexandros Potamianos, Petros Maragos, Konstantinos Rapantzikos, Georgios Skoumas, and Yannis Avrithis. Multimodal saliency and fusion for movie summarization based on aural, visual, and textual attention. *IEEE Transactions on Multimedia*, 15(7):1553–1568, 2013.

[52] Orhan Firat, Kyunghyun Cho, Baskaran Sankaran, Fatos T Yarman Vural, and Yoshua Bengio. Multi-way, multilingual neural machine translation. *Computer Speech & Language*, 2016.

[53] Carolina Galleguillos, Andrew Rabinovich, and Serge Belongie. Object categorization using co-occurrence, location and appearance. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[54] Haoyuan Gao, Junhua Mao, Jie Zhou, Zhiheng Huang, Lei Wang, and Wei Xu. Are you talking to a machine? dataset and methods for multilingual image question. In *Advances in neural information processing systems*, pages 2296–2304, 2015.

[55] Michael Gazzaniga, Richard B. Ivry, and George R. Mangun. *Cognitive Neuroscience: The Biology of the Mind*. Ww Norton and Co, 4th edition, 2016.

[56] Robert Gens and Pedro Domingos. Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems*, pages 3239–3247, 2012.

[57] Felix A Gers, Douglas Eck, and Jürgen Schmidhuber. Applying lstm to time series predictable through time-window approaches. In *Artificial Neural Networks ICANN 2001*, pages 669–676. Springer, 2001.

[58] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

[59] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[60] Michael Glodek, Stephan Tschechne, Georg Layher, Martin Schels, Tobias Brosch, Stefan Scherer, Markus Kächele, Miriam Schmidt, Heiko Neumann,

Günther Palm, et al. Multiple classifier systems for the classification of audio-visual emotional states. In *Affective Computing and Intelligent Interaction*, pages 359–368. Springer, 2011.

[61] Mehmet Gönen and Ethem Alpaydın. Multiple kernel learning algorithms. *Journal of machine learning research*, 12(Jul):2211–2268, 2011.

[62] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[63] Stephen Gould, Jim Rodgers, David Cohen, Gal Elidan, and Daphne Koller. Multi-class segmentation with relative location prior. *International Journal of Computer Vision*, 80(3):300–316, 2008.

[64] Abhinav Gupta, Aniruddha Kembhavi, and Larry S Davis. Observing human-object interactions: Using spatial and functional compatibility for recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(10):1775–1789, 2009.

[65] Mihai Gurban, Jean-Philippe Thiran, Thomas Drugman, and Thierry Dutoit. Dynamic modality weighting for multi-stream hmms inaudio-visual speech recognition. In *Proceedings of the 10th international conference on Multimodal interfaces*, pages 237–240. ACM, 2008.

[66] Thanh-Le Ha, Jan Niehues, and Alexander Waibel. Toward multilingual neural machine translation with universal encoder and decoder. *arXiv preprint arXiv:1611.04798*, 2016.

[67] Richard A Harshman. Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis. 1970.

[68] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[69] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[70] Geoffrey E Hinton and Terrence Joseph Sejnowski. *Unsupervised learning: foundations of neural computation*. MIT press, 1999.

[71] Frank L Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Studies in Applied Mathematics*, 6(1-4):164–189, 1927.

[72] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A field guide to dynamical recurrent neural networks*, 2001.

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

[74] Tao Hong, Pierre Pinson, and Shu Fan. Global energy forecasting competition 2012. *International Journal of Forecasting*, 30(2):357–363, 2014.

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

[76] Giridharan Iyengar, Harriet J Nock, and Chalapathy Neti. Audio-visual synchrony for detection of monologues in video archives. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, volume 5, pages V–772. IEEE, 2003.

[77] Andrew H Jazwinski. *Stochastic processes and filtering theory*. Courier Corporation, 2007.

[78] Siddharth Joshi and Stephen Boyd. Sensor selection via convex optimization. *IEEE Transactions on Signal Processing*, 57(2):451–462, 2009.

[79] Samira Ebrahimi Kahou, Xavier Bouthillier, Pascal Lamblin, Caglar Gulcehre, Vincent Michalski, Kishore Konda, Sébastien Jean, Pierre Froumenty, Yann Dauphin, Nicolas Boulanger-Lewandowski, et al. Emonets: Multimodal deep learning approaches for emotion recognition in video. *Journal on Multimodal User Interfaces*, 10(2):99–111, 2016.

[80] Suleiman A Khan and Samuel Kaski. Bayesian multi-view tensor factorization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 656–671. Springer, 2014.

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

[82] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[83] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[84] 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.

[85] Pieter M Kroonenberg and Jan De Leeuw. Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika*, 45(1):69–97, 1980.

[86] Zhen-zhong Lan, Lei Bao, Shoou-I Yu, Wei Liu, and Alexander G Hauptmann. Multimedia classification and event detection using double fusion. *Multimedia tools and applications*, 71(1):333–347, 2014.

[87] 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.

[88] Xinhai Liu, Shuiwang Ji, Wolfgang Glänzel, and Bart De Moor. Multiview partitioning via tensor methods. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):1056–1069, 2013.

[89] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[90] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[91] Cewu Lu, Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Visual relationship detection with language priors. In *European Conference on Computer Vision*, pages 852–869. Springer, 2016.

[92] Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

[93] Wolfgang Maass, Georg Schnitger, and Eduardo D Sontag. On the computational power of sigmoid versus boolean threshold circuits. In *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on*, pages 767–776. IEEE, 1991.

[94] Subhransu Maji, Lubomir Bourdev, and Jitendra Malik. Action recognition from a distributed representation of pose and appearance. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3177–3184. IEEE, 2011.

[95] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[96] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[97] Emilie Morvant, Amaury Habrard, and Stéphane Ayache. Majority vote of diverse classifiers for late fusion. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 153–162. Springer, 2014.

[98] Klaus-Robert Müller, Michael Tangermann, Guido Dornhege, Matthias Krauledat, Gabriel Curio, and Benjamin Blankertz. Machine learning for real-time single-trial eeg-analysis: from brain–computer interfacing to mental state monitoring. *Journal of neuroscience methods*, 167(1):82–90, 2008.

[99] Jun Nakanishi, Jay A Farrell, and Stefan Schaal. Composite adaptive control with locally weighted statistical learning. *Neural Networks*, 18(1):71–90, 2005.

[100] Ara V Nefian, Luhong Liang, Xiaobo Pi, Xiaoxing Liu, and Kevin Murphy. Dynamic bayesian networks for audio-visual speech recognition. *EURASIP Journal on Advances in Signal Processing*, 2002(11):783042, 2002.

[101] Ara V Nefian, Luhong Liang, Xiaobo Pi, Liu Xiaoxiang, Crusoe Mao, and Kevin Murphy. A coupled hmm for audio-visual speech recognition. In *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, volume 2, pages II–2013. IEEE, 2002.

[102] Natalia Neverova, Christian Wolf, Graham Taylor, and Florian Nebout. Moddrop: adaptive multi-modal gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(8):1692–1706, 2016.

[103] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011.

[104] Duy Nguyen-Tuong, Jan Peters, Matthias Seeger, and Bernhard Schölkopf. Learning inverse dynamics: a comparison. In *European Symposium on Artificial Neural Networks*, number EPFL-CONF-175477, 2008.

[105] Jianjun Ni, Xiaoping Ma, Lizhong Xu, and Jianying Wang. An image recognition method based on multiple bp neural networks fusion. In *Information Acquisition, 2004. Proceedings. International Conference on*, pages 323–326. IEEE, 2004.

[106] 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.

[107] 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.

[108] Dimitri Nion, Kleanthis N Mokios, Nicholas D Sidiropoulos, and Alexandros Potamianos. Batch and adaptive parafac-based blind separation of convolutive speech mixtures. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(6):1193–1207, 2010.

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

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

[111] Yaakov Oshman. Optimal sensor selection strategy for discrete-time state estimators. *IEEE Transactions on Aerospace and Electronic Systems*, 30(2):307–314, 1994.

[112] G. Pangalos, A. Eichler, and G. Lichtenberg. Tensor systems - multilinear modeling and applications. In *Proceedings of the 3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, pages 275–285, 2013.

[113] Evangelos E Papalexakis, Christos Faloutsos, and Nicholas D Sidiropoulos. Tensors for data mining and data fusion: Models, applications, and scalable algorithms. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):16, 2017.

[114] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 689–690. IEEE, 2011.

[115] Gerasimos Potamianos, Chalapathy Neti, Guillaume Gravier, Ashutosh Garg, and Andrew W Senior. Recent advances in the automatic recognition of audiovisual speech. *Proceedings of the IEEE*, 91(9):1306–1326, 2003.

[116] Vignesh Ramanathan, Congcong Li, Jia Deng, Wei Han, Zhen Li, Kunlong Gu, Yang Song, Samy Bengio, Charles Rosenberg, and Li Fei-Fei. Learning semantic relationships for better action retrieval in images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1100–1109, 2015.

[117] Geovany A Ramirez, Tadas Baltrušaitis, and Louis-Philippe Morency. Modeling latent discriminative dynamic of multi-dimensional affective signals. In *Affective Computing and Intelligent Interaction*, pages 396–406. Springer, 2011.

[118] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[119] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[120] Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

[121] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 635–644. ACM, 2011.

[122] Marcus Rohrbach, Wei Qiu, Ivan Titov, Stefan Thater, Manfred Pinkal, and Bernt Schiele. Translating video content to natural language descriptions. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 433–440, 2013.

[123] Frank Rosenblatt. Principles of neurodynamics. 1962.

[124] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[125] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual

Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[126] Mohammad Amin Sadeghi and Ali Farhadi. Recognition using visual phrases. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1745–1752. IEEE, 2011.

[127] Björn Schuller, Ronald Müller, Manfred Lang, and Gerhard Rigoll. Speaker independent emotion recognition by early fusion of acoustic and linguistic features within ensembles. In *Ninth European Conference on Speech Communication and Technology*, 2005.

[128] Luciano Serafini, Ivan Donadello, and Artur d'Avila Garcez. Learning and reasoning in logic tensor networks: theory and application to semantic image interpretation. In *Proceedings of the Symposium on Applied Computing*, pages 125–130. ACM, 2017.

[129] Amnon Shashua and Anat Levin. Linear image coding for regression and classification using the tensor-rank principle. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.

[130] Ali H Shoeb and John V Guttag. Application of machine learning to epileptic seizure detection. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 975–982, 2010.

[131] Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2017.

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

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

[134] Alex Smola and Vladimir Vapnik. Support vector regression machines. *Advances in neural information processing systems*, 9:155–161, 1997.

[135] Yale Song, Louis-Philippe Morency, and Randall Davis. Multi-view latent variable discriminative models for action recognition. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2120–2127. IEEE, 2012.

[136] Edwin Stoudenmire and David J Schwab. Supervised learning with tensor networks. In *Advances in Neural Information Processing Systems*, pages 4799–4807, 2016.

[137] 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.

[138] Shiliang Sun. A survey of multi-view machine learning. *Neural Computing and Applications*, 23(7-8):2031–2038, 2013.

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

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

[141] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

[142] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[143] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[144] Volker Tresp, Cristóbal Esteban, Yinchong Yang, Stephan Baier, and Denis Krompaß. Learning with memory embeddings. *arXiv preprint arXiv:1511.07972*, 2015.

[145] Volker Tresp and Yunpu Ma. The tensor memory hypothesis. *NIPS 2016 Workshop on Representation Learning in Artificial and Biological Neural Networks (MLINI 2016)*, 2016.

[146] Volker Tresp, Yunpu Ma, Stephan Baier, and Yinchong Yang. Embedding learning for declarative memories. In *European Semantic Web Conference*, pages 202–216. Springer, 2017.

[147] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International Conference on Machine Learning*, pages 2071–2080, 2016.

[148] Yan-Hui Tu, Jun Du, Qing Wang, Xiao Bao, Li-Rong Dai, and Chin-Hui Lee. An information fusion framework with multi-channel feature concatenation and multi-perspective system combination for the deep-learning-based robust recognition of microphone array speech. *Computer Speech & Language*, 46:517–534, 2017.

[149] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[150] Endel Tulving. Episodic memory: From mind to brain. *Annual review of psychology*, 53(1):1–25, 2002.

[151] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[152] Victoria Uren, Philipp Cimiano, José Iria, Siegfried Handschuh, Maria Vargas-Vera, Enrico Motta, and Fabio Ciravegna. Semantic annotation for knowledge management: Requirements and a survey of the state of the art. *Web Semantics: science, services and agents on the World Wide Web*, 4(1):14–28, 2006.

[153] M Alex O Vasilescu and Demetri Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In *European Conference on Computer Vision*, pages 447–460. Springer, 2002.

[154] M Alex O Vasilescu and Demetri Terzopoulos. Multilinear image analysis for facial recognition. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 2, pages 511–514. IEEE, 2002.

[155] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010, 2017.

[156] Sethu Vijayakumar and Stefan Schaal. Locally weighted projection regression: An o (n) algorithm for incremental real time learning in high dimensional space. In *International conference on machine learning, proceedings of the sixteenth conference*, 2000.

[157] Sethu Vijayakumar and Stefan Schaal. Locally weighted projection regression: An o (n) algorithm for incremental real time learning in high dimensional space. In *International conference on machine learning, proceedings of the sixteenth conference*, 2000.

[158] Weiran Wang, Raman Arora, Karen Livescu, and Jeff Bilmes. On deep multi-view representation learning. In *International Conference on Machine Learning*, pages 1083–1092, 2015.

[159] Qi Wu, Damien Teney, Peng Wang, Chunhua Shen, Anthony Dick, and Anton van den Hengel. Visual question answering: A survey of methods and datasets. *Computer Vision and Image Understanding*, 163:21–40, 2017.

[160] Yi Wu, Edward Y Chang, Kevin Chen-Chuan Chang, and John R Smith. Optimal multimodal fusion for multimedia data analysis. In *Proceedings of the 12th annual ACM international conference on Multimedia*, pages 572–579. ACM, 2004.

[161] Zhiyong Wu, Lianhong Cai, and Helen Meng. Multi-level fusion of audio and visual features for speaker identification. In *International Conference on Biometrics*, pages 493–499. Springer, 2006.

[162] Huaxin Xu and Tat-Seng Chua. Fusion of av features and external information sources for event detection in team sports video. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 2(1):44–67, 2006.

[163] Huijuan Xu and Kate Saenko. Ask, attend and answer: Exploring question-guided spatial attention for visual question answering. In *European Conference on Computer Vision*, pages 451–466. Springer, 2016.

[164] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In David Blei and Francis Bach, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2048–2057. JMLR Workshop and Conference Proceedings, 2015.

[165] Zenglin Xu, Feng Yan, et al. Infinite tucker decomposition: Nonparametric bayesian models for multiway data analysis. *arXiv preprint arXiv:1108.6296*, 2011.

[166] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

[167] Bangpeng Yao and Li Fei-Fei. Grouplet: A structured image representation for recognizing human and object interactions. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 9–16. IEEE, 2010.

[168] Bangpeng Yao and Li Fei-Fei. Modeling mutual context of object and human pose in human-object interaction activities. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 17–24. IEEE, 2010.

[169] Guangnan Ye, Dong Liu, I-Hong Jhuo, and Shih-Fu Chang. Robust late fusion with rank minimization. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3021–3028. IEEE, 2012.

[170] Özgür Yilmaz, Artur S d'Avila Garcez, and Daniel L Silver. A proposal for common dataset in neural-symbolic reasoning studies. In *NeSy@ HLAI*, 2016.

[171] Rose Yu and Yan Liu. Learning from multiway data: Simple and efficient tensor regression. In *Proceedings of the 33nd International Conference on Machine Learning (ICML-16)*, pages 238–247, 2016.

[172] Hanwang Zhang, Zawlin Kyaw, Shih-Fu Chang, and Tat-Seng Chua. Visual translation embedding network for visual relation detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, page 4, 2017.

[173] Jixian Zhang. Multi-source remote sensing data fusion: status and trends. *International Journal of Image and Data Fusion*, 1(1):5–24, 2010.

[174] Hua Zhou, Lexin Li, and Hongtu Zhu. Tensor regression with applications in neuroimaging data analysis. *Journal of the American Statistical Association*, 108(502):540–552, 2013.