Karlsruhe Institute of Technology

Topic Prediction in Dialogs using Convolutional Neural Networks

Master's Thesis of

Kuangyuan Jin

At the Department of Informatics
Institute for Anthropomatics and Robotics

Reviewer:	Prof. Dr. Alexander Waibel
Second reviewer:	Prof. Dr. Tamim Asfour
Advisor:	Dr. Jan Niehues

Duration: April 26, 2017 – October 25, 2017

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, October 25, 2017

.....

(**Kuangyuan Jin**)

Zusammenfassung

Dialog State Tracking ist ein unverzichtbarer Bestandteil in einem modernen Sprachdialogsystem, der die Handlungen des Systems direkt beeinflusst. Für nicht zielorientierte Dialogsysteme ist das Gesprächsthema ein Dialog State von hohem Interesse. Um eine passende Antwort zu generieren muss das System nicht nur das aktuelle Thema verfolgen können, sondern auch Thema für den darauffolgenden Zug vorhersagen. Das Ziel dieser Arbeit ist es, einen Ansatz zu implementieren, der die drei Aufgaben Topic Tracking, Next Topic Prediction und Topic Change Prediction lösen kann. Hierfür werden drei unterschiedliche Convolutional Neural Network Architekturen verwendet und zwar ein Grundmodell, ein mehrschichtiges Modell und ein Word Embedding Modell. Techniken zur Optimierung wie Dropout, L1 und L2 Regularisation werden eingesetzt, um die Modelle gegen Overfitting zu stärken. Da nur nicht gelabelte Dialogdaten verfügbar sind, die zu den vorgegebenen Aufgaben passen, werden diese manuell mit passenden Topic Labels annotiert, um ein Dialogkorpus zum Trainieren und Testen der Netze zu erhalten. Außerdem werden mehrere Wortmodelle zur Generierung von Vektorrepräsentationen trainiert, die verwendet werden um die Gesamtperformance zu verbessern.

Die Modelle werden umfangreich bezüglich mehreren Modell- und Trainingsparametern ausgewertet, um die jeweils besten Werte zu finden. Die beste Performance wird mit dem Grundmodell erreicht und zwar mit einem Cross-validation F-score von 0,750 für Tracking, 0,652 für Next Topic Prediction und 0,818 für Topic Change Prediction. Obwohl die F-scores der Prediction Aufgaben relativ hoch sind, scheint es, dass Vorhersagen für Themenwechsel immer noch eine sehr schwierige Aufgabe sind. Insgesamt ist es dem Modell für Topic Prediction gelungen, 31% der neuen Themen und 56% der Themenwechseln korrekt vorherzusagen. Trotz den Fehlern kann die Performance der Netze als gut betrachtet werden, da Themenvorhersage in Gesprächen eine Aufgabe ist, die sogar für Menschen sehr schwierig ist. Außerdem sind die Modelle für beide Prediction Aufgaben in den meisten Fällen genau, wenn kein Themenwechsel stattfindet.

Abstract

Dialog state tracking is an essential component in a modern spoken dialog system which directly influences the actions of such a system. For non-goal-oriented dialog applications, the conversation topic is a dialog state of high interest. In order to generate appropriate answers, the system has to be able to not only track the current topic but also make topic predictions for the subsequent turn. The goal of this work is to implement an approach to solve the three tasks of topic tracking, next topic prediction and topic change prediction. Therefore we employ three different architectures of convolutional neural networks which are a basic model, a multilayer network and a word embedding network. Optimization techniques such as dropout, L1 and L2 regularizations are applied to strengthen our models against overfitting. Since only unlabeled dialog data are available which are suited for our tasks, we manually annotate the data with appropriate topic labels to create a dialog corpus for training and testing our networks. Additionally, we also train several word models to generate vector representations which are employed to improve the overall performance on our given tasks.

Our models are extensively evaluated on various model and training hyperparameters to find optimal values for each of them. The best performance is achieved by the basic model with a cross-validation F-score of 0.750 for tracking, 0.652 for next topic prediction and 0.818 for topic change prediction. Although the scores obtained for the prediction tasks are relatively high, it seems that making predictions for topic changes is still a very difficult task. Our topic prediction models are able to correctly predict 31% of the new topics and 56% of all topic changes. Despite the errors, we think that the performance of our networks can still be considered as good since making topic predictions in conversations is a task that is difficult even for human. Furthermore, the models for both prediction tasks are mostly accurate when no topic change takes place.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

Spoken dialog systems are computer applications which are able to interact with users in a natural language. Compared to alternative systems where the user has to enter commands manually, spoken dialog systems allow a hands- and eyes-free communication which can be essential in many cases. For example, when a car driver wants to interact with the on-board computer, it can be very dangerous if the driver is distracted from a manual input. However, with a computer system that can communicate with voice, the driver can concentrate on the traffic while giving instructions to the computer. Speech as input is also a much faster way of communication and is very natural and easy to use.

In recent years, conversational systems are gradually becoming a part of daily life, with examples including Apple's Siri, Google Now and Microsoft's Cortana. The main components of a typical spoken dialog system can be found in figure 1.1.

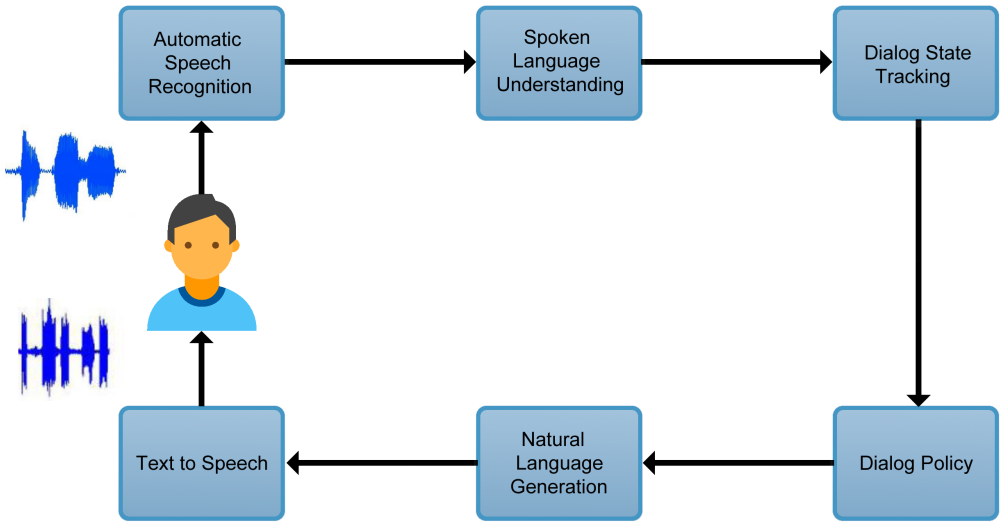

Figure 1.1: Main components of a modern spoken dialog system.

The first component is the automatic speech recognition (ASR) which interacts with the user and receives the spoken utterances as audio data. Its task is to decode the sound waves and transform them into a text form. The output of the ASR is then passed to the spoken language understanding (SLU) component which aims to derive meaningful representations out of the recognized utterances. The SLU component is followed by a dialog state tracker which makes estimations of the current dialog state using the SLU results. This new dialog state is passed to the dialog policy which decides on an appropriate action depending on the updated dialog state. The system's response to the user is then created by the natural language generation component which converts the output of the dialog policy into texts in a natural language. Finally, the generated texts are transformed into an audio form using the text to speech component and are then passed back to the user.

This work focuses on dialog state tracking which is a difficult task because ASR and SLU errors are common due to noises in the input audio and ambiguities in natural languages. Errors in previous steps can cause the system to misunderstand the user and significantly influence the performance of the tracking component. At the same time, a robust dialog state tracker is essential because the dialog policy completely relies on the estimated dialog state to choose the right actions. Unlike the tracking component illustrated in figure 1.1, which uses the SLU output to determine the dialog state, there are also trackers that employ the ASR output instead or outputs from both the ASR and SLU. Our approach is based on the ASR results alone, which means that raw user utterances are directly processed to make estimations for the dialog state.

Non-goal-oriented dialog systems are applications that aim to build natural conversations with the users without trying to accomplish any goals or tasks. For these systems, the current topic of the conversation contains important information that needs to be maintained and hence can be seen as part of the dialog state. It is essential for the system to keep track of the topics in order to continue the conversation. Tracking in non-goal-oriented dialogs is usually more difficult than in goal-oriented dialogs since the user's utterances are not as specific and may contain statements that do not provide a lot of information. Most of the previous works on dialog state tracking are employed for goal-oriented systems and to our knowledge, there are no well-known approaches for tracking topics in social conversations to date. Additionally, tracking topics enables the system to follow the conversation, but a good dialog policy component is still needed to generate appropriate answers. A component, which not only tracks the topic of the current turn but is also able to generate topic predictions for the subsequent turn, contains functionalities of the dialog policy component as well which in some cases can be omitted. This combination of tracking and dialog policy is also a field which has not been studied well and introduces the possibility to novel approaches.

Over the years, many techniques have been developed to solve the task of dialog state tracking in goal-oriented systems, including rule-based methods, statistical machine learning and neural network approaches. Convolutional neural networks are models inspired by the biological visual system which are capable of learning from examples. Although they are primarily developed for image classification tasks, they have demonstrated excellent performance on tasks involving natural languages. On the field of dialog state tracking, they have been successfully employed as well and

are able to achieve state-of-the-art performance. Following the success of previous works, we apply convolutional neural networks and evaluate them on the new tasks of topic tracking and prediction in non-goal-oriented dialogs.

1.2. Overview

This work is organized as follows. First, chapter 2 provides the necessary basics on convolutional neural networks, including the algorithm for network training and different optimization techniques. It also formalizes the dialog state tracking problem and introduces a text pre-processing scheme which converts words into vector representations. Chapter 3 reviews several interesting works using convolutional neural networks for different kinds of applications. Various related researches on the field of dialog state tracking are also presented and a comparison of these works and our own tasks can be found at the end of the chapter. Our own approach is described in detail in chapter 4 along with a precise definition of our tasks. Chapter 4 also includes the data creation process for the labeled dialog corpus used for training and the three different network architectures we employ to solve our given tasks which are the basic model, the multilayer network and the word embedding network. Chapter 5 provides a further analysis of our data and introduces the metrics used to evaluate the performance of our models. It then covers the most important results achieved during our experiments including those from thorough hyperparameter tuning, implementation of regularization schemes and evaluation of the different network architectures. At the end of the chapter, we present the final results achieved by our best models. Finally, chapter 6 concludes by summarizing the most important aspects of this work and proposes ideas for possible future works.

2. Basics

In recent years, machine learning algorithms have rapidly become popular due to the large amount of data that today's applications need to process. Artificial neural networks are computing systems inspired by biological neural networks in order to solve problems in the same way that a human brain would. With their ability to derive patterns from imprecise data, they can be trained to learn trends from given examples and apply their knowledge in real world systems. Neural networks have been successfully employed for various kinds of tasks where the problems are too complex for conventional algorithmic methods. They have demonstrated excellent results on tasks involving natural language, therefore we also implement a neural network approach to solve our given tasks on transcribed spoken dialogs. This chapter provides a brief introduction to the field of neural networks, along with the convolutional architecture employed in this work. It also gives an explicit definition of dialog state, a general term used to represent information in dialogs which the topic of a conversation belongs to. Additionally, we employ a technique to create continuous representations of words in order to improve the performance of our neural network model. A description of this scheme can be found in the following sections as well.

2.1. Neural Networks

Neural networks were primarily developed back in 1943 by the scientists McCulloch and Pitts in [McPi43] to describe how neurons in the brain might work. Their paper inspired various researches on the field of artificial neural network like the computational machines in [FaCl54] and [RHHD56], and the perceptron in [Rose58]. The first functional networks with many layers were published in [IvLa67], but research on this field stagnated after [MiPa69] because computers at that time did not have enough processing power to effectively handle large models. With hardwares growing more and more powerful, a renewed interest in neural networks led to many key advances like further development of the backpropagation algorithm introduced in [Werb75], deep-learning on a large scale through the use of GPUs and various model architectures like recurrent networks in [Schm93] and restricted Boltzmann machines

in [Smol86]. As of today, deep neural networks have won various international competitions like handwriting recognition in [GrSc09] and traffic sign recognition in [CMMS12], with the latter achieving human-competitive or even superhuman performance. For a better understanding of the neural network approach employed in this work, a brief overview of the standard feedforward network as well as the more powerful convolutional architecture is given in the following. To train the network, the backpropagation algorithm is used which is described in this section as well, along with strategies to optimize the result of the training.

2.1.1. Feedforward Neural Networks

A simple neural network consists of a collection of units, the neurons, and the connections between them. Neurons are processing devices that take several inputs to produce a single output. The importance of the respective inputs to the output is expressed by weights and the units usually also have an activation function which determines the final output of the neuron. Given the inputs $x_1, x_2, ..., x_n$, the respective weights $w_1, w_2, ..., w_n$ and the activation function f, the neuron's activated output o can be calculated by

$$o = f(\sum_{i=1}^{n} x_i w_i) = f(\mathbf{x} \cdot \mathbf{w}).$$

The weighted sum of the inputs can be expressed as the inner product of the input vector and the weight vector as seen on the right side of the equation. The activation function is a nonlinear function which is important for neural networks to solve problems that are not linear. Commonly used activation functions are the threshold, the sigmoid and the hyperbolic tangent function. By connecting the outputs of certain neurons to the inputs of other ones, a directed and weighted network is formed.

Typically, neurons are organized in a layered architecture depending on their interactions with the environment and the different kinds of transformations they perform on their inputs. Neurons that receive inputs from the outside form the input layer and those that produce outputs to the outside are part of the output layer. Units

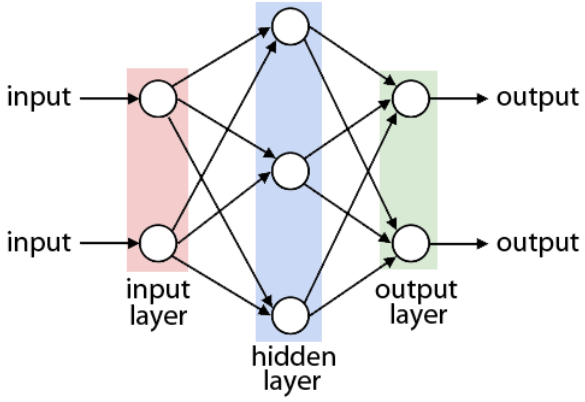

Figure 2.1: A standard feedforward neural network with input and output layer, and one hidden layer.

that are only connected to other neurons are hidden from the environment and thus belong to the hidden layers. Figure 2.1 shows a simple neural network consisting of an input and an output layer with two neurons each and one hidden layer with three units. Since the directed connections only go from a previous layer to the next one and not backwards, it is called a feedforward network. It is also a fully-connected model because the output of each neuron in one layer is connected with every neuron of the subsequent layer. Besides the standard feedforward network there are more complex architectures like recurrent networks and convolutional networks which are developed to perform different tasks.

2.1.2. Convolutional Neural Networks

Convolutional neural networks (CNNs) are special feedforward models which are inspired by the receptive fields of the animal visual system and the strong spatially local correlation in natural images. Variations of the CNN architecture are for example the time delay networks described in [WHHS+89]. Similar to the ordinary network illustrated in figure 2.1, CNNs also consist of an input and an output layer, as well as one or multiple hidden layers. The hidden layers are either convolutional, pooling or fully-connected. Typically, the input layer is a convolutional layer and the output layer is fully-connected to its previous layer. A convolutional layer is usually followed by a pooling layer and fully-connected hidden layers are mostly placed towards the end of the network. Since CNNs are often used for image classification, the layers are sometimes arranged in three dimensions reflecting the input's width, height and color depth.

Convolutional layers are sparsely connected to the input or the previous layer as each neuron in a convolutional layer is only connected to a small region of the preceding layer. The connection weights between such a region and a neuron are shared across the entire input, so a different neuron connected to a different region of the input also has the same set of weight parameters. All neurons of a convolutional layer that share the same parameters are grouped together and form a feature map. Mathematically, the outputs of a feature map are computed by applying a convolution operation to the input with the shared weights as the filter. Given the input matrix $\mathbf{X}$ of size $M_x \times N_x$ and a 2-dimensional filter $\mathbf{W}$ of size $M_w \times N_w$, the convolution is expressed with $\mathbf{C} = \mathbf{X} * \mathbf{W}$ where each element of the output is calculated by

$$\hat{\mathbf{C}}(i,j) = \sum_{m=0}^{M_w-1} \sum_{n=0}^{N_w-1} \mathbf{X}(i-m, j-n) \cdot \mathbf{W}(m,n)$$

where $0 \leq i < M_x + M_w - 1$ and $0 \leq j \leq N_x + N_w - 1$. Note that this definition also performs convolution on the outer bound of the input where the filter area exceeds the input. Since this is not the case in convolutional layers, the actual output of a feature map is the slice of $\hat{\mathbf{C}}$ without the border results

$$\mathbf{C} = \hat{\mathbf{C}}((M_w - 1){:}(M_x - 1), (N_w - 1){:}(N_x - 1))$$

where : indicates the sequence slice between the starting index on the left and the stopping index on the right. Convolutional layers use multiple sets of filter weights

and mostly consist of a large number of feature maps. But through weight sharing, they usually still have much fewer free parameters than fully-connected layers of similar size. Since the filter sizes are comparatively small, their complexity is solely given by the number of feature maps. The collection of all feature map outputs can be taken as the final output of the convolutional layer, but typically, an activation function is further applied to ensure the nonlinearity of the layer.

Another important concept of CNNs is the pooling layer, which is often employed after a convolutional layer. Pooling is a form of nonlinear downsampling and can be performed on one entire feature map or on a small cluster of inputs. Therefore the input is partitioned into a set of regions which are defined by the pooling size and stride. One output is generated for each region, for example by taking the maximum or the average value. Figure 2.2 illustrates a non-overlapping max-pooling process with a pooling size of 2×2.

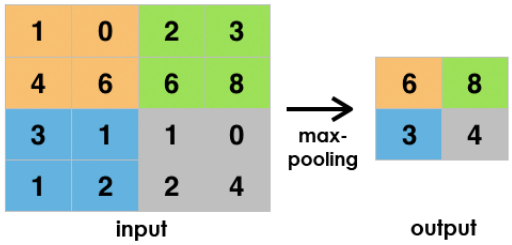

Figure 2.2: Example of 2-dimensional max-pooling with a pooling size of 2×2 and a stride of 2. Source: [1].

Pooling aims to reduce computation for subsequent layers by removing values that do not have a high importance. It is also able to process inputs with different sizes and return outputs of the same size for subsequent layers which only work on one fixed input size. Additionally, a form of translation invariance is provided, since small translations of the input would not have a very big impact on the output. The convolutional and pooling layers are often followed by fully-connected layers to increase the computational power of the model and to convert the network output into a classification decision. A shallow CNN architecture usually employs one convolutional and pooling layer while a deep model consists of multiple pairs of convolutional and pooling layers and one or two fully-connected hidden layers.

2.1.3. Network Training

Neural networks as classifiers are superior to rule-based algorithms because they are able to learn patterns and trends from examples. Rather than being specifically programmed for a given task, they can be trained to solve different kinds of problems depending on the input data. Training neural networks employs algorithms that gradually modify the network's parameters, for example the weights on the connections between each two neurons. During the training process, the network is given a desired output or a loss function which it tries to reproduce or to minimize through adjustments to its parameters. A set of model parameters that yields the most favored result is usually found after iterating several times over the entire training

[1] https://en.wikipedia.org/wiki/Convolutional_neural_network

data. These parameters can then be saved and employed to solve the same task on new data. Various learning algorithms are available for neural networks and even with a very efficient algorithm, training a large model can take a great amount of time. Hence it is often necessary to reduce model complexity as a balance between the model's performance and the effort to train it.

A common method to train neural networks is the backpropagation algorithm which was originally introduced in the 1970s. But its importance was not fully appreciated until the paper [RuHW86] by Rumelhart et al. was published. The authors found out that backpropagation performed much faster than earlier learning approaches, making it possible to use neural nets to solve problems which had previously been insolvable. Backpropagation has been essential to the growth in neural network research over the years and even today, it is still the most successful and widely used algorithm for training. It can be employed on a wide range of different network architectures like the standard feedforword models as well as the CNN architectures used in this work. Because of the reduced number of parameters, training CNNs using backpropagation is usually even more efficient than standard feedforward models.

The backpropagation algorithm, which is also called backpropagation of errors, consists of two major steps, propagation and weight update. When the input is processed by the model, information is propagated forward through the network until it reaches the output layer. The output of the network is then compared to the desired output and an error value is calculated for each of the neurons in the output layer. Network training is carried out by minimizing the number of errors the network commits and therefore an error function, also called loss function, is needed to compute the error. An intuitive loss function is the zero-one loss which simply returns 1 if the classification result is correct and else 0. But since the goal is to optimize the model with regards to the error function and therefore we need to compute the gradient, the non-differentiable zero-one loss is rather inappropriate. Thus, maximizing the log-likelihood of the model given a training set, or here minimizing the negative log-likelihood, is a more common approach. The negative log-likelihood NLL is defined as

$$NLL(\theta, D) = -\sum_{i=1}^{N} \log f_\theta(x_i) = -\sum_{i=1}^{N} \log P_\theta(Y = y_i|x_i)$$

where θ is the set of all model parameters and D the training dataset consisting of N example pairs. The input values and label of example i are denoted by x_i and y_i, respectively. The continuous likelihood function f_θ is a discrete distribution and can be expressed by the term $P_\theta(Y = y_i|x_i)$. It describes the probability of the correct label y_i given the input x_i which is computed by the network during the forward propagation. P_θ can be directly taken from the output of the softmax function as it already yields a probability distribution over the label classes.

After calculating the error value, it is then propagated backwards starting from the output layer. To minimize the error for each output neuron and the network as a whole, each of the parameters in the network has to be updated according to its influence on the overall error. Given a model parameter θ_i and we want to know how much a change in θ_i affects the total error, it is appropriate to calculate the gradient of the loss function with respect to this parameter

$$\Delta\theta_i = \frac{\partial NLL(\theta, D)}{\partial \theta_i}.$$

The next step is to update the parameters so that the error becomes as small as possible. This can be achieved using the gradient descent algorithm. It is a method to find a local minimum of the loss function by repeatedly taking small steps downward on the error surface. The steps taken by one model parameter are proportional to the negative of the gradient with respect to this parameter, so its new value $\hat{\theta}_i$ is calculated by

$$\hat{\theta}_i = \theta_i - \gamma \Delta\theta_i$$

where γ is the learning rate, a constant which defines the length of the steps made in direction of the negative gradient. The learning rate is a hyperparameter which is chosen and optimized through experiments.

Once the parameters are updated, a new iteration is started by generating model outputs, evaluating the error and calculating the gradients. In this way, the network parameters are adjusted iteratively in order to find a set of parameter values that minimize the error. Instead of the standard gradient descent algorithm, its stochastic variant is often used which estimates the gradients directly from one example at a time instead of the entire training set. This simplifies the gradient calculation and hence proceeds more quickly than the original method. The whole training algorithm consisting of backpropagation and stochastic gradient descent is summarized in algorithm 1.

Algorithm 1 Backpropagation with Stochastic Gradient Descent

```
1: Initialize network parameters θ with random values
2: while looping do
3:     for each example (x_i, y_i) in training set do
4:         Generate network output using model parameters θ
5:         Compute probability P_θ(Y = y_i|x_i)
6:         Compute loss function NLL(θ, x_i, y_i)
7:         Compute gradient Δθ_k for all parameters
8:         Update parameter θ_k by −γΔθ_k for all parameters
9:     end for
10:    if <stopping condition> then
11:        Set looping to false
12:    end if
13: end while
14: return parameters θ
```

While bias parameters can be simply initialized with 0, weights are usually set to small numbers randomly generated from a symmetric interval. Glorot and Bengio suggested in [GlBe10] a combination of the fan-in and fan-out

$$r = \sqrt{\frac{6}{\text{fan-in} + \text{fan-out}}}$$

where fan-in and fan-out are the number of inputs and outputs of the unit, respectively. Weights are then uniformly sampled from the interval $[-r, r]$. The training algorithm stops when a specified maximum number of iterations is reached or when the model has converged. This means that there were no significant improvements to the model during the last few iterations and hence suggests that a local minimum was found. The speed of convergence depends highly on the chosen learning rate γ used for gradient descent. A larger value for the learning rate accelerates the training speed but there is also the possibility that the steps made are too big to find the minimum. It is necessary to experiment with various learning rates to optimize the training algorithm. For further information on backpropagation and stochastic gradient descent, see [LBOM12] and [LeBH15] for example.

2.1.4. Optimizations

Deep neural networks become more powerful with growing complexity and number of model parameters. But with the increasing size of the networks, they also become more vulnerable to overfitting, which is one of the biggest challenges on the field of machine learning. Overfitting is a serious problem that occurs during training. It is caused by the network trying to minimize the error on the training data too much and hence memorizing the training examples instead of learning classification patterns. An overfitted model does not generalize well which means it performs badly on examples which were not part of the training data. One way to combat overfitting is to divide the training data and keep two sets of examples exclusively for validation and testing purposes which we also apply in our work. While the testing set is used to evaluate the performance of the network on unseen data, the validation set is employed to determine whether to continue training or to stop before it converges. If the model's performance ceases to improve sufficiently on the validation set, or even degrades with further optimization, then it is an indication of the network starting to overfit and that no more training is needed. To further improve the generalization ability of our CNN models, three other optimization schemes are employed additionally. These three methods are L1 regularization, L2 regularization and dropout which are described in the following sections.

2.1.4.1. L1 and L2 Regularization

L1 and L2 regularization, sometimes also called as weight decay, are commonly used techniques to reduce overfitting. The idea of L1 and L2 regularization is to add an extra term to the loss function, a term called the regularization term. It is defined as the sum of the absolute values of the model weights for the L1 regularization and the squared sum of the weights for the L2 regularization

$$L1 = \sum_{w} |w|, \quad L2 = \sum_{w} w^2$$

where w iterates over all weight parameters from the convolutional, the hidden and the output layer. For the loss function from section 2.1.3

$$NLL(\theta, D) = -\sum_{i=1}^{N} \log P_\theta(Y = y_i | x_i)$$

the regularized loss function becomes

$$E(\theta, D) = NLL(\theta, D) + \lambda_1 * L1 + \lambda_2 * L2$$

with the hyperparameters λ_1 and λ_2 which determine the relative importance of the two regularization terms. Both λ_1 and λ_2 are commonly small numbers since the original loss should still be the most significant term to minimize. Adding the regularization terms to the loss function encourages the model to learn small weights which results into simple network solutions. Large weights are mostly penalized and only allowed if they considerably improve the first part of the regularized loss function.

L1 and L2 regularization can be viewed as a way of compromising between finding small weights and minimizing the original loss function. When the network is able to find simple solutions, it does not necessarily mean that it also generalizes well. But empirically, it was found that performing such regularization helps with reducing the effects of overfitting, especially on small datasets. One explanation is that if a weight vector is very large, it most likely stays pointing in the same direction, since a change due to gradient descent only makes a very small difference to the direction of a long vector. A different point of view states that with small weights, changes in the input do not have a big impact on the behavior of the model. This makes the regularized network less prone to noises in the data and it learns mostly from examples which are often seen across the training set. Since regularizing bias parameters does not change the result very much empirically, they are usually not included in the L1 and L2 regularization terms.

2.1.4.2. Dropout

An almost certain way for performance improvement is to combine many models that either have different architectures or were trained on different data. But training different architectures is expensive and there are usually not enough data available. Dropout is a regularization technique that not only helps to improve generalization but also provides a way of approximately combining many different network architectures efficiently. Unlike L1 and L2 regularization, dropout does not modify the loss function, but the network itself.

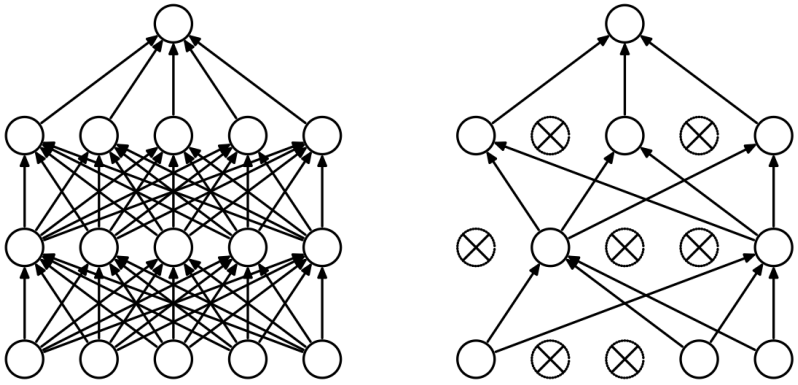

Figure 2.3: Dropout in a standard neural network with fully-connected layers. Source: [SHKS+14].

The idea is to drop units with a fixed probability during training which means that they are temporarily removed from the network, along with all their incoming and outgoing connections. Figure 2.3 illustrates this process in a standard neural network with fully-connected layers. The left image shows the model before dropout and the right shows a thinned model produced by applying dropout to it. This scheme prevents the units from overfitting too much as it breaks up co-adaptions of feature detectors since the dropped out units cannot influence other retained units. Additionally, a new thinned network is sampled for each training iteration consisting of all the units that remained after the dropout. So training a neural network with dropout can be seen as a very efficient form of model averaging where the number of different models is exponential in that of the units and these models share the same parameters.

Dropout can be conveniently implemented by applying a binary mask on the input of the layer. Given the original input $\mathbf{x}$ and the mask $\mathbf{m}$, the input after dropout $\hat{\mathbf{x}}$ is calculated with

$$\hat{\mathbf{x}} = \mathbf{x} \circ \mathbf{m}$$

where $\circ$ denotes element wise product and $\mathbf{m}$ is a vector of the same length as $\mathbf{x}$ with entries drawn independently from a Bernoulli distribution with the given dropout probability. For CNN models, dropout can not only be employed in fully-connected layers, but also in the convolutional and pooling layers. Applying this technique on the convolution output feature map turns the subsequent max-pooling into a stochastic process.

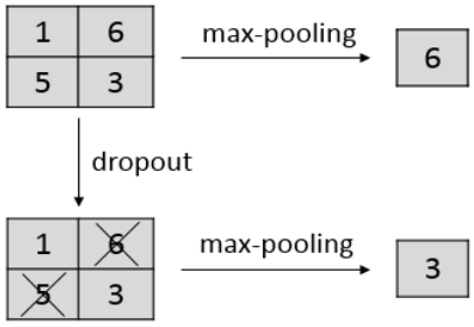

Figure 2.4: Example of applying dropout before max-pooling on an output feature map of size 4. Source: [WuGu15].

An example showing the procedure of max-pooling dropout is depicted in figure 2.4. Without dropout, the strongest activation 6 is always selected as the output. But with dropout, each unit in the pooling region could be removed which leads to possible different pooling results. In this case, since the actual maximum 6 has been dropped out, 3 is chosen as the pooled output instead. Max-pooling dropout functions as a randomized activation according to the dropout probability distribution and also increases the number of possibly trained networks. Experiments confirm that dropout before max-pooling further improves the performance which was observed in [WuGu15]. More details on dropout regularization can be found in [SHKS+14] and [HSKS+12].

2.2. Dialog State

Dialog systems are computer applications that are developed to converse with human in a natural language. To build a conversation, it is important for the computer to track what has happened in the dialog and to respond according to the context. As the dialog progresses, the dialog system needs to maintain a representation of the previous and current content of the conversation which is generally called the dialog state. Depending on the application, there are different kinds of conversation content that can be defined as part of a dialog state. For example when a user interacts with a travel planning system to search for a hotel, the dialog state might indicate the search parameters for the type of hotel such as the desired star rating, location and price range. A non-goal-oriented dialog system on the other hand usually needs to follow the topic of the conversation in order to generate sensible replies to the user's utterances.

Dialog state tracking is the task to determine representations for a current dialog state and to update them at each moment for an on-going conversation in a spoken dialog system. Since the output of the dialog state tracking component is used to decide what action should be taken next, it is an essential component in dialog systems and therefore has a huge impact on the final performance of the complete system. Spoken dialogs usually have a rich information content, hence it is usually required to track a set of multiple values and maintain them as the dialog state. But since dialog systems often have a specific purpose, it is only necessary to track dialog states that are relevant to the given purpose. For systems that help a user to complete a task, dialog state tracking is typically applied to the user's goal and other information required to fulfill the goal. Possible values for dialog states need to be determined for the specific domain beforehand in order to apply them during the tracking process. These values, also referred as the ontology, specify the scope of what the system understands and the tasks that it can help the user complete.

A definition of slot-based dialog state tracking was given in the Dialog State Tracking Challenge which is an on-going series of research community competitions for developing state-of-the-art dialog state trackers (see 3.2.1 for further information on the challenge). This definition was firstly applied to human-computer dialogs with systems that help the user search an entity of a database with the user's desired attributes. According to the definition, the dialog state can be represented by a set of slots which need to be filled with values. These slots can be divided into two types of subsets, the constraint slots and the request slots. Constraint slots are attributes of the database entities that the user may use to limit their search by specifying a value. Request slots are attributes that might also be of interest but the user does not, in many cases cannot, specify a value for them.

Examples for constraint and request slot labels can be found in the dialog excerpt in table 2.1. Here, the interaction between a user and a dialog system for restaurant queries is illustrated where a dialog state is defined for every turn of the conversation consisting of one utterance per speaker. For each slot, a set of possible values is defined and used to describe the state of the dialog segments. Constraint slot values like the attributes area and price range are explicitly specified by the user and hence consist of a name and value pair. Phone number and address, which are attributes that the user asks for, belong to the request slots and fill them with the name of

Speaker	Utterance	Dialog State
Computer:	Which part of town?	Constraint slots:
User:	A cheap place in the North.	area=*north*, pricerange=*cheap*
		Request slots:
		-
Computer:	Clown café is a cheap restaurant in the North part of town.	Constraint slots: area=*south*, pricerange=*cheap*
User:	Do you have any others like that, maybe in the South part of town?	Request slots: -
Computer:	Galleria is a cheap restaurant in the South.	Constraint slots:
User:	What is their number and address?	area=*south*, pricerange=*cheap*
		Request slots:
		phone, *address*

Table 2.1: Example dialog excerpt between an user and a dialog system for restaurant queries annotated with the according constraint and request slot values. Source: [HeTW14a].

the attribute only. In case the user has no preference for a constraint slot, it is still treated as a constraint and a ”dontcare” value is assigned to the attribute. The two slot subsets are not necessarily disjoint, for example the user might constrain the search by stating a desired price range or ask for it given a search result. For human-computer dialogs as shown in the example, dialog states can commonly be defined for every turn since each of the user's utterances usually contains enough information to fill the slots. In human-human conversations, this is usually not the case and hence a dialog state is often defined over a span of multiple turns. Since our task also involves conversation between real persons, we use dialog segments instead of single turns. More information on how dialog states are determined in this work can be found in section 4.1.

2.3. Distributed Representation of Words

CNNs can be directly trained on raw dialog text data with minimum preprocessing and simple encoding schemes. Word encodings can for example be created by building a vocabulary containing all occurring words and representing the words with their respective indices in the vocabulary. But this scheme causes the word representations to loose information about similarities between them, hence words should not be treated as simple discrete indices, but also the relationships that may exist between the words should be provided. For similar words like 'dog' and 'cat', their representations should be close as well, so the model can use what it has learned about 'dogs' when it processes the word 'cat'. To create word representations that can replicate word similarities faithfully, it is necessary to increase the complexity of these representations, i.e. using vectors instead of scalars and continuous values instead of discrete ones. Finding good word embeddings can be an important step since it is possible that the overall performance of the CNN models can be improved in this way as well.

There are publicly available methods that are able to generate embeddings of words in a continuous vector space. A well-known and efficient tool for learning word

embeddings is `word2vec` [2]. This scheme was initially introduced in [MCCD13] and was further improved in [MSCC+13]. `Word2vec` models are shallow neural networks which learn to map words into a low-dimensional vector space from their distributional properties observed in an input text corpus. They are able to group the vectors of semantically similar words together to nearby points in the vector space. There are two distinct models, the continuous bag-of-words (CBOW) model and the skip-gram model. These models are trained to predict context and target words, for example given the word context 'dogs and cats are' and the target word 'pets', the CBOW model tries to predict the word 'pets' from the context, while the skip-gram model does the inverse and predicts the context words from the target word. Using either the CBOW or the skip-gram model, word vector representations are trained which can be subsequently employed in many natural language processing applications.

A well trained `word2vec` model can accurately make guesses about a word's meaning based on its occurrences in the given training data. Those guesses can be used to establish a word's association with other words which can be investigated by calculating the distance between the word vectors. By entering a word, the `distance` function will display the most similar words and their distances to the specified word. For example for the word "france", the output should look like

```
       spain      0.678515
     belgium      0.665923
 netherlands      0.652428
       italy      0.633130
 switzerland      0.622323
  luxembourg      0.610033
    portugal      0.577154
      russia      0.571507
     germany      0.563291
   catalonia      0.534176
         ...
```

with a top 10 consisting of mostly other major European countries. Interesting linguistic regularities can also be observed by performing arithmetic operations on the word vectors. For example by computing $\mathbf{v}_{\text{paris}} - \mathbf{v}_{\text{france}} + \mathbf{v}_{\text{italy}}$ which indicate the word vector for Paris, France and Italy, respectively, the resulting vector is one that is very close to $\mathbf{v}_{\text{rome}}$. To observe such strong regularities in the word vector space, it is needed to train the models on a very large data set and with sufficient vector dimensionality.

A Python adaption of the original C tool is implemented in the Python library gensim which is a collection of various tools for topic modeling, document indexing and similarity retrieval tasks. Detailed information on the gensim framework can be found in [ŘeSo10]. Additional to core `word2vec` functionalities, gensim also provides useful methods such as preparing the input and building a vocabulary. Gensim's `word2vec` implementation takes a text corpus as input and yields the trained model as output which can be saved and loaded as a Python dictionary consisting of word

[2] https://code.google.com/archive/p/word2vec/

vector pairs for each word in the vocabulary. For the input there are special sentence iteration methods for the Brown corpus [3] and the Wikipedia text8 corpus [4], as well as a general method for texts consisting of one sentence per line. These sentences can be directly used as input for training the word model, but to further improve the training data, gensim's `phrases` module can be applied beforehand. The `phrases` function is able to automatically detect common multiword expressions from a stream of sentences. For example if the input stream contains the phrase "new york", it will detect it as a collocation and convert it into a single word token "new_york". Using `phrases`, the `word2vec` model is able to learn more meaningful multiword expressions instead of single words.

After preparing the input data, the vector space word model can be trained using gensim's `word2vec` method. This method accepts several parameters that affect both training speed and quality. For example a minimum count of a word can be specified, so words that rarely appear in a very large corpus can be treated as typing errors and removed from the dictionary. The dimensionality of the feature vectors can also be defined, as well as the number of iterations the model is trained on the input data. It is also possible to choose between the CBOW and skip-gram model and set the number of threads for training parallelization. The trained model can be stored and loaded with the standard gensim methods which use Python's pickle module internally. A persisted model can be loaded from file into a dictionary object for word vector lookup operations or to continue training with more sentences. Similar to the original C tool, gensim provides various word similarity tasks that can be performed with the model, for example finding the most fitting words given positives and negatives or a mismatch in a stream of words. Overall, gensim's `word2vec` is a very useful tool for implementing machine learning in Python which not only ported the original training algorithm, but also extended it with additional functionalities for natural language tasks.

[3] http://www.hit.uib.no/icame/brown/bcm.html
[4] http://mattmahoney.net/dc/textdata

3. Related Work

With the growing availability of training data, methods based on convolutional neural networks have been successfully employed in many new technologies, especially when the tasks are too difficult for traditional computer algorithms. CNNs can be applied on a wide range of applications like image classification, video recognition and natural language processing. Results of various kinds of competitions show that they are able to achieve state-of-the-art performance and outperform several other machine learning approaches. This chapter presents an overview of interesting works using CNNs with focus on text and image classification tasks. It also provides a summary of previous works on the field of dialog state research where different kinds of techniques are briefly introduced. Many recent works on this field were developed during one of the Dialog State Tracking Challenges. An introduction to these series of challenges can also be found in the following sections.

3.1. CNN Applications

CNNs are commonly employed for image processing tasks since they require relatively little pre-processing compared to other image classification algorithms. They are able to learn the filters by themselves where for traditional algorithms, prior knowledge and human effort are needed to design the filters. Standard neural networks have also been successfully used for image recognition, but due to the full connectivity between neurons, they suffer from the exponential increase of weight parameters and do not scale well to higher resolution images. Other than image classification, CNNs have also been successfully applied on areas like video recognition, information filtering and natural language processing. The following sections provide an overview of various application areas where CNNs have been able to achieve excellent results.

3.1.1. ImageNet Classification

ImageNet is a huge dataset consisting of 15 million high resolution images containing one object each. These images are hand-labeled with an object label from over 22,000 object classes. A competition is held every year since 2010 which is titled

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) where one task is to classify the objects of the ImageNet dataset. For the 2010 competition a subset of the database containing 1.2 million images and 1000 object classes was used. Participating systems had to generate five labels ordered by their probabilities for each input image. Performance was tested using two metrics which are the top-1 and top-5 error rates evaluating how often the correct label was not among the best one and best five labels, respectively. More details on ILSVRC can be found in [BeDFF10] and [RDSK+14].

Krizhevsky et al. trained a large, deep convolutional neural network to classify the images in the ILSVRC-2010 contest in their work [KrSH12]. Their CNN consisted of five convolutional layers, some of which were followed by max-pooling layers, and three fully-connected layers with the last one being a 1000-way output layer. An illustration of the entire architecture can be found in figure 3.1.

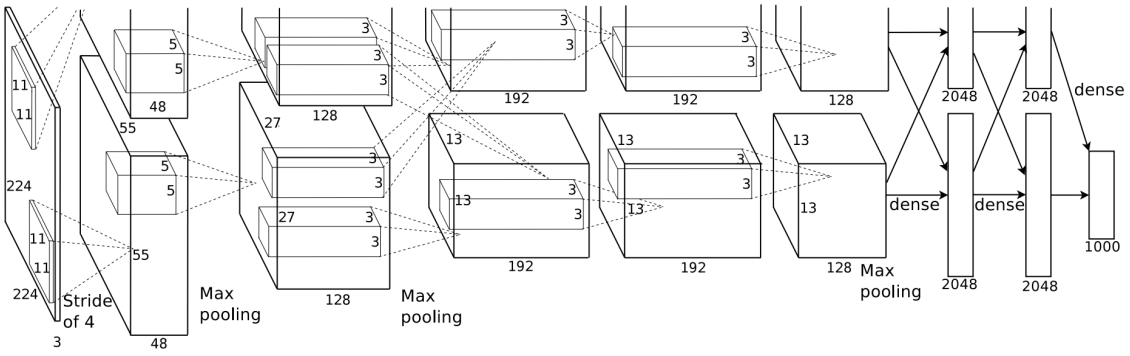

Figure 3.1: Architecture of the CNN employed for the large scale object classification task. The numbers denote the 3-dimensional sizes of the input, the layers and the filters, respectively. Source: [KrSH12].

Overall, the CNN had about 650,000 neurons and 60 million parameters with most of parameters coming from the fully-connected layers. The network was divided into two parts between the first and the last layer which means that it was trained on two GPUs, since a single GPU did not have enough memory for a CNN of this size. Using two GPUs not only enabled training very large networks but also helped to reduce the error rates. To further improve the performance of their network, the authors employed various optimizations on their architecture like rectified linear units which are neurons that use the maximum function as activation. Regularization techniques like the local response normalization and dropout were applied as well to improve the generalization ability of the model. For the pooling layers, an overlapping pooling scheme was used where the pooling regions overlap with each other.

On the test data, the CNN achieved top-1 and top-5 error rates which were considerably better than the previous state-of-the-art. With an error rate of 37.5% for top-1 and 17.0% for top-5, it surpassed the former best result of 45.7% and 25.7% by 9.6% and 11.2% absolute, respectively. Krizhevsky et al. also entered a variant of their model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% by the second-best entry. Their undisputed victory marked a turning point for large-scale object recognition in the history of ILSVRC and their influence could be clearly seen in the subsequent competitions. Following their success, the vast majority of entries employed deep CNNs in their

submissions for the contest. The winner of the ILSVRC-2013 classification task used the average of several large deep convolutional networks which is described in [ZeFe14] and achieved a top-5 error rate of 11.7%. Even more significant progress was made during ILSVRC-2014 with a top-5 error rate of 6.7% by [SLJS+14]. CNN approaches were also able to win other ILSVRC task categories, for example the entries from [SEZM+13], [SiZi14b] and [LiCY14].

3.1.2. Text Classification

Text classification is a common topic on the field of natural language processing where the task is to assign a text document to predefined categories. For this task, traditional algorithms are usually used which are based on simple statistics of some ordered word combinations like n-gram models. Zhang et al. employed deep CNNs for text classification in [ZhZL15] which operated on texts at character level. They compared their approach to traditional models and their character-level CNNs were able to achieve state-of-the-art or competitive results.

The architecture of their CNNs is illustrated in figure 3.2. It consisted of 6 convolutional layers, each followed by a max-pooling, and 3 fully-connected layers. A large and a small CNN were trained where the large model contained 1024 feature maps in each of its convolutional layers and 2048 units in the fully-connected hidden layers. As input, the model took a sequence of character vectors which were generated from the input text through 1-of-n encoding.

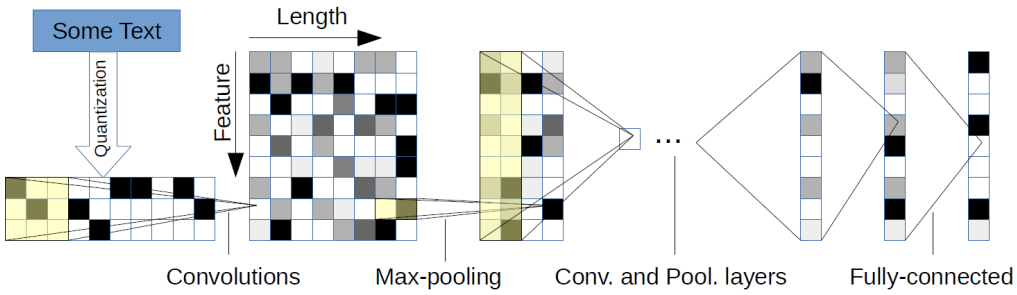

Figure 3.2: Architecture of the character-level CNN employed for text classification. Source: [ZhZL15].

Since CNNs usually need a great amount of training data to perform well, Zhang et al. built several large-scale datasets for their experiments with several millions of samples, mostly from news articles and reviews on the internet. To further enhance the training of their network, they experimented data augmentation by using an English thesaurus obtained from Word-Net [Fell05]. For all of the large datasets, their CNN models were able to outperform traditional methods like bag-of-words and n-gram models. With smaller datasets of size up to several hundreds of thousands, the CNNs performed slightly worse than the best models. Their experiments demonstrated that CNNs for text classification can be successfully employed on character-level given sufficient amount of data.

Other works on the field of text classification are for example [Kim14] where simple CNNs with little hyperparameter tuning were trained for sentence-level classification tasks. These CNNs were able to achieve excellent results on multiple benchmarks. In [JoZh14], instead of low-dimensional word vectors as input, high-dimensional

text data were directly applied to CNNs to exploit the structure of texts. A semi-supervised framework with CNNs was presented in [JoZh15] which was able to learn embeddings of small text regions from unlabeled data. To counter the ambiguity problems in short texts because of their lack of context, a method based on CNN and semantic clustering was introduced in [WXXL+15]. For a detailed analysis of how CNN architecture components effect the overall model performance in text classification tasks, see [ZhWa15].

3.1.3. Other Applications

Aside from ImageNet, CNNs have been successfully employed on other image datasets like the MNIST database of handwritten digits. An error rate of 0.23% on this database was reported in [CiMS12] where deep CNNs with winner-takes-all neurons were trained to solve this task. On the MNIST handwriting benchmark, this approach was the first to achieve near-human performance and on a traffic sign recognition benchmark it even outperformed humans by a factor of two. Excellent results for object classification on the datasets NORB and CIFAR10 were obtained in [CMML+11] with fast trained CNN variants. They achieved error rates of 2.53% and 19.51% on NORB and CIFAR10, respectively. For the considerably more difficult face recognition task, CNNs were able to significantly reduce error rate compared to alternative methods as stated in [LGTB97]. In [MMMK03], a 97.6% recognition rate was reported on 5,600 still images of more than 10 subjects. A deep CNN demonstrated in [Tech15] the ability to spot faces from a wide range of angles with competitive performance, even when the faces are partially occluded. CNNs have also been explored on the field of video analysis which is much more complex than images because of the additional temporal property. The paper [CaVGB06] describes a CNN application for automatic and objective measurement of the quality of digital videos. Action recognition in videos as well has been experimented with deep CNNs in works like [JXYY13], [KTSL+14] and [SiZi14a].

CNN models have also been proven to be effective on tasks involving natural language. Like text classification, CNNs achieved excellent results on the field of semantic parsing which was shown in [GBFH14]. The authors stated that their approach was especially suitable to grammatically malformed or syntactically atypical texts. The paper [SHGD+14] presented semantic models based on CNNs to learn low-dimensional semantic vectors for search queries and web documents. Results demonstrated that the novel model significantly outperformed other semantic models, which were the prior state-of-the-art. Deep CNN models could also be used on sentiment analysis of short texts as shown in [SaGa14]. The authors achieved state-of-the-art results on the Stanford Sentiment Tree-bank for single sentence sentiment prediction in both binary classification and fine-grained classification tasks. A dynamic CNN was described for the semantic modelling of sentences in [KaGB14]. The network achieved excellent performance in sentiment prediction tasks and over 25% error reduction with respect to the strongest baseline. In the paper [SaZa14], a deep CNN was proposed that was able to learn character-level representation of words to perform Part-of-Speech tagging. Using the proposed approach, Part-of-Speech taggers for English and Portuguese were produced which achieved over 97% accuracy on the respective corpora. Further works on CNNs on the field of natural language processing can be found in [CoWe08], [CWBK+11] and [ZhLe15].

3.2. Dialog State Tracking

Numerous techniques for dialog state tracking have already been proposed prior to the Dialog State Tracking Challenges. But due to different domains, system components and evaluation metrics that were used, a direct comparison of progresses on this field was not possible. Ever since a common testbed was provided by the first competition as well as labeled dialog data, a variety of methods have been developed and successfully applied to dialog state tracking. While during the first challenge, less than half of the entries were able to outperform the simple baselines, results of the last two challenges demonstrated great advances as most of the submissions were able to outperform the given baselines. This section provides an overview of the challenges as well as various techniques developed over the years from older works to recent state-of-the-art approaches.

3.2.1. Dialog State Tracking Challenge

The Dialog State Tracking Challenge (DSTC) is an on-going series of competitions held since 2013. The task for each of the past challenges was to create a tracker that can determine the dialog state for new dialogs. Labeled dialog data, a common testbed and evaluation framework were provided for the challenges to support the participating research teams. Simple baseline trackers were implemented each time as a comparison for the submissions. For all competitions, the dialog corpus used was taken from a specific domain and the conversations were between two speakers with one asking for information and the other providing that information. While the first three competitions had focus on human-machine dialogs, DSTC4 and DSTC5 differed from the previous challenges by employing human-human dialogs as training and test data. Over the years, DSTCs have enabled the development of new methods for dialog state tracking, as well as various evaluation techniques.

Since there were no common evaluation measures for the task of dialog state tracking prior, the goal of the first DSTC was to provide such a testbed to enable progress comparison on this field. A corpus of 15,000 transcribed and labeled human-computer dialogs between users and a bus schedule information system was used to train and test the trackers. The outputs of the trackers were evaluated using 11 metrics and compared to two simple baselines. Details on the evaluation metrics and the competition results can be found in [WRRB13]. Similar to the first challenge, DSTC2 and DSTC3 studied the problem of dialog state tracking on human-computer dialog corpora. Instead of a system for bus information, the second and third challenge each used data from the restaurant search domain. While only constraint slots were employed during DSTC1, the dialog states were extended by including request slots and search methods identified in the user's utterances. The performance of the participating models during these two subsequent competitions were thoroughly analyzed and the results demonstrated the difficulty of the problem. An overview of DSTC2 and DSTC3, and a summary of the results can be found in [HeTW14a] and [HeTW14b], respectively.

From the fourth challenge onwards, the focus changed from human-computer dialogs to human-human dialogs. The goal of the main task in DSTC4 was to develop trackers for conversations between tourists and tour guides. A corpus consisting of 35 dialog sessions was used which contained touristic information for Singapore

and summed up to over 31,000 utterances. Since dialog states could not always be expressed in just a single turn in these conversations, they were defined for dialog segments spreading over multiple turns. In addition to the main task, the challenge included a series of optional pilot tasks for developing end-to-end dialog systems using the same dataset. For a detailed description on DSTC4, see the paper [KDBW+16a]. Following a similar scheme to DSTC4, the fifth edition again used dialogs in the tourist information domain, but introduced a cross-language dialog state tracking task. Therefore trackers for a target language had to be built using existing resources in the source language and the corresponding machine translated sentences in the target language. Like DSTC4, different pilot tasks were proposed for the core components in developing end-to-end dialog systems following the same cross-language setting. A summary of the results achieved during DSTC5 can be found in [KDBW+16b].

3.2.2. Elaborate Rule-based Tracker

With the increased size of the ontology and the utterances labeled at a subdialog level during DSTC4, a dialog state tracking method designed to work robustly under these new conditions was introduced in [DLBB16]. In this work, an elaborate tracker was constructed which combined multiple rule-based techniques. The whole system was implemented in a pipeline structure and figure 3.3 shows the four main steps of the pipeline.

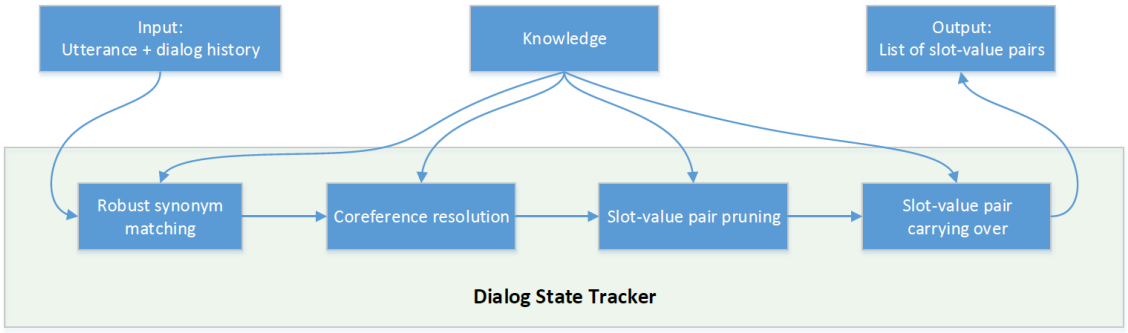

Figure 3.3: Pipeline structure of the elaborate rule-based tracker with four main steps. Source: [DLBB16].

In addition to the current utterance and the dialog history, which is a list of previous utterances and predicted dialog states, the systems also received knowledge of the problem domain in form of the ontology as input. The first step in the pipeline performed a keyword matching between the input and the ontology to detect slot value pairs. In order to enhance the robustness of the detection, a synonym list of the ontology was used as well which for example included plural forms of nouns in the ontology. After finding slot-value pairs by matching synonyms, the tracker further applied a resolution system for place-related coreferences. Using different templates, it aimed to resolve indirect place-related statements, e.g. ”our hotel”, and fill the slot with the exact name of the place. Since the outputs from the previous steps often resulted in closely related slot values with only one being a correct dialog state label, the tracker utilized the domain knowledge in the ontology as well as observations from the training data to select the most likely slot-value pair during the third step. Finally, the detected slot-value pairs were carried over to the following utterances

until another value appeared because many slot-value pairs were observed to remain present for several subsequent utterances. A hybrid tracker was developed as well which used the rule-based tracker's outputs and combined it with a machine learning based classifier.

The performance of the elaborate approach was evaluated on the DSTC4 test set on both utterance-level and subdialog-level in comparison to the baseline and submissions by other research groups. Both the rule-based and the hybrid tracker were able to yield performances far above the fuzzy matching baseline. For the utterance-level evaluation, the rule-based model and the hybrid model achieved the best F-scores of 0.52 and 0.53, respectively. Only one other team was able to obtain a score of 0.45 and all other team were below 0.35 which reflected the difficulty of the task. Similar for the subdialog-level, almost all teams obtained F-scores of less than 0.40, while both the rule-based and hybrid trackers were able to achieve a score of 0.57 which outperformed the second best entry at a score of 0.50. The results of DSTC4 demonstrated that elaborate rule-based approaches are able to yield competitive results, when the size of the ontology is large and the utterances are labeled at the subdialog-level only. It was also shown that further performance improvement can be obtained by combining both rule-based and machine learning approaches.

3.2.3. Multichannel CNN Tracker

A purely machine learning based approach was presented in [SUEY+17c] to solve the new cross-language dialog state tracking task introduced in DSTC5 where trackers had to be built on an English training corpus and evaluated on an unlabeled Chinese corpus. The authors proposed a multichannel CNN architecture in which English and machine translated Chinese were treated as different input channels of one single CNN model. The overall architecture of the multichannel CNN can be found in figure 3.4.

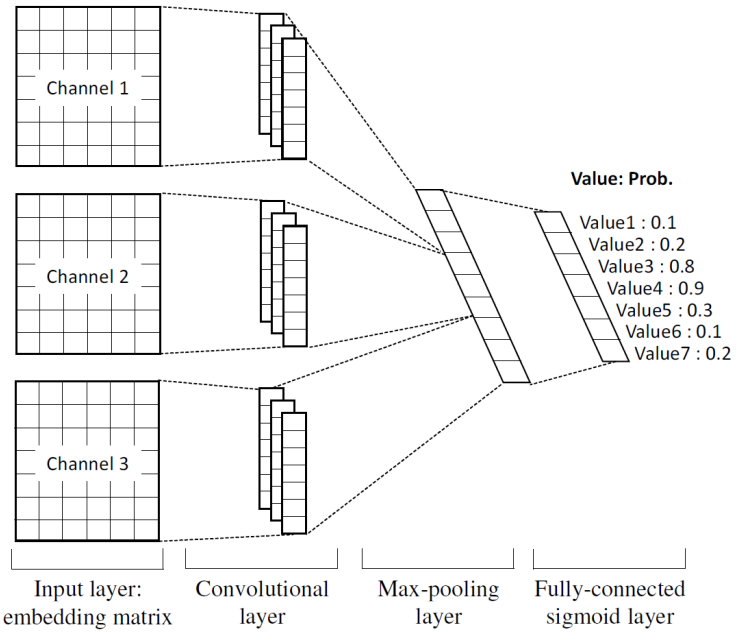

Figure 3.4: Architecture of the multichannel CNN model with three input channels for cross-language dialog state tracking. Source: [SUEY+17c].

The input of each channel is a 2-dimensional matrix representing the input text using word embeddings. Two different embeddings were generated for Chinese and one for English, hence the full model input consisted of three channels. Different sets of filters were used for each channel because word embeddings among different languages varied greatly and thus should not be filtered in the same way. The authors trained five models with different number of filters ranging from 1200 for the smallest to 2800 for the largest network. They also implemented the dropout technique to reduce overfitting and experimented with different dropout probabilities for each of the models.

For the DSTC5 evaluation, the multichannel CNN model was able to achieve the best score among all participating teams. Their best model outperformed the second best team by 50% relative with an accuracy of 0.0956 compared to 0.0635. A 15% improvement from 0.3945 to 0.4519 could be achieved when using the F-measure as evaluation metric. The multichannel model was found to be robust against translation errors and outperformed monolingual single channel CNN models. The CNN based approach was not only able to achieve best performance, but also did not require any prior knowledge about the target languages and therefore can be easily applied to other languages.

3.2.4. Other Methods

Since dialog state tracking is an important component in a dialog system, various techniques have been developed over the years. Early systems mostly used rule-based approaches, for example the framework for dialog managers built in [LaTr00] where a set of information state variables was tracked using hand-written update rules. A probabilistic approach was introduced in [HiNA03] which aimed to resolve speech ambiguity based on heuristic scores obtained from dialog corpora. Bayesian networks were also applied to spoken dialog systems as shown in [WiYo07] and [DeSt07] where they demonstrated significant performance gains compared to existing techniques. In the paper [BoRu06], a belief updating model which tracked speech understanding hypotheses was presented and numerous variations of this approach could be found in subsequent works like [MeBW13] and [Will14].

For the first DSTC, many different approaches could be found among the submissions, including hand-crafted rules [WaLe13], decision trees [Will13] and deep neural networks [HeTY13]. The dialog state tracker proposed in [LeEs13] was the winning entry at DSTC1 which was built upon a combination of techniques like a wide-coverage data selection, a feature-rich discriminative model and unsupervised prior adaptation. The word-based tracking method presented in [HeTY14b] was among the winning teams for the different tasks of DSTC2. It was based on a recurrent neural network structure which required very little feature engineering and demonstrated consistently high performance across all of the tasks and metrics. Other DSTC2 submissions employed for example linear conditional random fields [KiBa14] and a combination of deep neural networks and maximum entropy models [SCZY14b]. Similar approaches could also be found in the submissions for DSTC3 as well as rule-based methods proposed in [KVLM$^+$15] and [SCZY14a], and a novel dialog state tracker based on a Markovian neural network model [ReXY14]. The recurrent neural network model which achieved great results during DSTC2 could be further improved in [HeTY14a] and was also able to obtain best scores in many

DSTC3 tasks. Aside from the elaborate rule-based tracker and the multichannel CNN described in 3.2.2 and 3.2.3, there were various other approaches submitted to the DSTC4 and DSTC5 with most of them outperforming the baseline by a large margin. A system combining probabilistic and rule-based strategies was introduced in [LiWu17] and [SuLW16] for the fourth and fifth challenges, respectively. Works like [SUEY+17a], [YHNN17] and [JHLC+16] employed neural network architectures to solve the given tasks. For further details on dialog state tracking approaches and more information on DSTC evaluations, see [Hend15] and [WiRH16].

3.2.5. Discussion

Although a large number of techniques has been developed over the years, those methods have in common that they only aim to track the current state of the dialog and do not make any decisions. For a complete dialog system to utilize the output dialog states, further components are required to evaluate them and take actions depending on the evaluation result. Our task is to expand the idea of dialog state tracking and combine it with a decision making capability for the subsequent step of time. This has, to our knowledge, not yet been addressed in any existing works and hence is a novel approach on the field of dialog state research. The majority of previous works are also employed in a context where the data corpus is taken from a specific domain and the speakers are divided into someone who requests information and someone who provides that information. In contrast to these goal-oriented dialog systems, this work aims to develop techniques for non-goal-oriented, social dialog systems in which the speakers discuss about everyday topics and are equivalent conversation partners.

As seen in the history of DSTC, neural network approaches, especially CNNs, have been able to achieve state-of-the-art performance on the task of dialog state tracking. Compared to other methods like hand-written rules, they have the advantage that no prior domain or language knowledge is required to build such a tracker. For training a CNN classifier for dialog tasks, it is usually not required to build very deep models as shallow CNNs can be successfully employed as well. Therefore, we apply different variations of the CNN architecture to track dialog states relevant to our task and additionally train models that are capable of making predictions for subsequent dialog states.

4. Convolutional Neural Networks for Topic Prediction

Tasks involving spoken dialogs are challenging because natural language is often complex, ambiguous and depends mostly on the speech context. Traditional pattern matching methods have been the state-of-the-art on this field previously, but those approaches often ignore the contextual information in dialogs and are not able to capture the whole semantics in the words. Hence, as seen in section 3.2, they could be outperformed by the latest neural learning approaches. Convolutional Neural Networks (CNNs) are often used for image and video classification tasks, but they also show great results when applied to natural language processing systems. In this work, three variations of CNN architectures are employed to accomplish the given tasks. The following sections contain a detailed description of each of the three network architectures as well as an explicit definition of our given tasks. For training the networks, it is necessary to collect a dialog corpus that fits the task specifications. Since only not labeled dialog data are publicly available which fit the criteria, the appropriate topic labels need to be added manually. A description of the data creation process can also be found in this chapter.

4.1. Task Definition

The goal of this work is to create CNN models which are able to determine representations of topics in human-human dialogs. Our tasks involve identifying the dialog state in an ongoing conversation, as described in 2.2. Since the models in this work are developed for non-goal-oriented dialog systems, the relevant dialog states differ from those for a database related system. A user interacting with such a system does not try to achieve some goal, so there are no constraining attributes but rather informative ones. Informative slots can be treated in the same way as constraint slots and filled with name value pairs. For our purpose of predicting topics during a conversation, we define ”topic” as an attribute for the informative slots. There are also other attributes that can been seen as relevant to the dialog state, e.g. a more detailed representation of what the speakers have said. But for this work, we limit the scope to the attribute ”topic” and treat it as the only part of dialog state that

we are interested in. Hence instead of "dialog state", we simply use the word "topic" in the following.

Besides predicting the next topic in a conversation, it is also of interest to track the current topic and to predict the occurrence of a topic change. Therefore, we train networks on three different tasks which are tracking, next topic prediction and topic change prediction. A great advantage is that the same model architecture and learning algorithm can be employed for any of the three tasks. To make the networks learn features for different tasks, it is only necessary to alter the way the training data is used. For the tracking task, the training example pairs consist of a dialog segment and its corresponding topic label, so the model is trained to classify the current state of the dialog. The input for the two prediction tasks is the same as for the tracking task, but different reference output labels are used. For the next topic prediction task, the topic label of the dialog segment, which directly follows the current one, is used as the reference output. For the topic change prediction task, the reference output is set to 1 if the next topic label differs from the current and else to 0. Training neural networks on these task variant inputs will automatically lead the models to learn solutions for the different tasks without the need of further adjustments.

4.2. Data Creation

As described in the previous section, our tasks are not only to classify the current state of the dialog but also to predict topic changes in the conversation flow. Therefore we need dialog data where the speakers do not only talk about a single topic but frequently change the conversation subject. The tasks further specify that the network models should work on non-goal-oriented conversations, also known as social dialog. The transcribed Switchboard-1 [1] dialog corpus available at the KIT Interactive Systems Lab [2] unfortunately could not be employed for this work, because the dialogs discuss about one specific subject and no significant topic changes could be found. A few other unlabeled dialog corpora were available on the internet, including the Cornell Movie-Dialogs corpus [DNMLe11], the Ubuntu Dialog corpus [LPSP15] and the TRAINS Dialog corpus [3]. But the final choice fell on the Talkbank Conversation corpus because of the huge size of the text collection and its relevance for the specifications.

Talkbank [4] is a large collection of various kinds of multilingual recordings and transcriptions which support fundamental research in the study of human-human communication. It was established by the Carnegie Mellon University in cooperation with the University of Pennsylvania and contains sample databases for several subfields of communication research, including language acquisition, conversation analysis and medical science. These databases are publicly shared and collected in order to advance the development of standards and tools for creating systems for linguistic purposes. Further information and a detailed description of all available Talkbank databases can be found in [MacW07].

[1] https://catalog.ldc.upenn.edu/LDC97S62/
[2] http://isl.anthropomatik.kit.edu/english/
[3] https://catalog.ldc.upenn.edu/LDC95S25/
[4] http://talkbank.org/

The data used in this work are taken from Talkbank's BilingBank database which is part of the CABank, a collection of dialog corpora for conversation analysis research. It contains over 150,000 lines of daily conversations which were recorded by Universities from around the world like the USA, England, Argentina, Singapore and many more. There is a great diversity in the group of speakers as it ranges from teenagers, who talk about things like school and hobby, to elderly people who often speak of health. The recorded conversations are transcribed by researches from the respective universities where information about the speakers like gender, age and the relationship between them were also included. As the name implies, the speakers in these corpora are bilingual and sometimes switch between English and their native language. It is desired to work on an English vocabulary only and fortunately the transcribers also included a translation for parts that were not spoken in English. Aside from the translations, the transcribed texts also contain additional information like part of speech tags, special words for sounds and comments by the researchers. Since the main purpose of the BilingBank corpora is to analyze speech patterns and code-switching, there are no labels or tags related to the contents of the conversations. Hence, we need to manually label as much data as possible with the corresponding topics to create a dataset for training our networks.

Before starting the labeling, the dialogs have to be cleaned up because all the additional information tags make the texts hard to read. After removing the not required tags and replacing the non-English parts with the respective translations, we can finally annotate the dialogs with appropriate labels. Since it is hard to find topics on an utterance-level, labels are thus defined for dialog segments of multiple utterances. Therefore the dialogs are divided into segments consisting of 3 turns and only conversations between 2 speakers are selected which means that a dialog segment always has the length of 6 utterances. Making the dialog segments equal in length simplifies further processing steps and a size of 3 turns seems to be appropriate as it is long enough to represent a dialog state. Due to the limited time of this work, only a subset of the BilingBank corpora can be labeled. Also there are some parts in the dialogs where no specific conversation topic can be determined or the context of the statement is unclear and those parts are removed. This is done carefully, so no changes to the conversation flow are made.

There were no specific topics given, so the topic labels have to be found during the labeling process. While reading through the dialogs segment by segment, we intuitively assign an appropriate topic label to the dialog segment which we consider to be a good generic term representation for the current conversation subject. The labels found in this way are collected in an ontology, so if we want to assign a certain topic label to a dialog segment which is not already part of the ontology, it is then added to the list. Since the conversations are informal, the resulting topics are all common subjects of everyday life like school or work. Some dialog segments also contain a topic change within and those are assigned the topic label which spans over more utterances and thus has more importance in that segment. After the labeling process, the initial ontology consists of 30 topics, but since there are topic labels with close semantics and not many examples each, they are fused together to a single topic label. The final ontology contains 23 distinct topic labels and the full list can be found in section 5.1. An exemplary extract from the labeled training corpus annotated with the corresponding topic labels can be found in table 4.1. In this example, the first and third segments can be unambiguously labeled with

Speaker	Utterance	Dialog State
*TIM:	Right now I have to go to the university because I'm going to register for some classes.	
*MIG:	Oh.	
*TIM:	Plus now I have to take a whole bunch of classes.	Topic=*Education*
*MIG:	How many classes?	
*TIM:	Three.	
*MIG:	How many were you taking?	
*TIM:	Dunno. Well, now I have to study all day.	
*MIG:	No kidding, then when you look the years have gone by and you'll be way behind.	
*TIM:	Exactly. So I'm going to register for the classes and I'm going to study, I don't know, I think maybe all day.	Topic=*Education*
*MIG:	And you're not going to work.	
*TIM:	I don't think so.	
*MIG:	Why?	
*TIM:	Because no, I don't really have to work.	
*MIG:	Well, once you start working it's like, even if you stop working and even if they're paying everything, it's like, I don't know.	
*TIM:	Yeah, man. In any case, I'm going to get a part time job at the weekends.	Topic=*Work*
*MIG:	Right, a little part time job there, some extra dough there, does you no harm at all.	
*TIM:	Yeah.	
*MIG:	Mhm.	

Table 4.1: Sample dialog segments from the training corpus annotated with corresponding topic labels.

the topics *Education* and *Work*, respectively. A topic change is indicated in the second segment and since the focus here is still on the previous topic, it also receives *Education* as label.

4.3. Models

To solve the given tasks, we implement a CNN approach which has not only been successfully applied on tasks involving images, but also on text classification tasks. Researches have demonstrated that CNNs work well on raw data with minimal amounts of preprocessing. They can also be directly applied without any knowledge on the syntactic or semantic structures of a language which is a major advantage compared to rule-based methods. Additionally, the convolution operation helps to reduce the number of free parameters and improve the generalization ability of the model. Three variations of CNN architectures are employed in this work to accomplish the given tasks. They differ in their composition of layers and the amount of input data preprocessing, but have the same form of output. The architecture of each of the three networks is presented in detail in the following sections.

4.3.1. Basic Model

The network models used in this work are based on Kim's architecture for sentence classification as described in [Kim14]. The author was able to achieve excellent

results on multiple benchmarks with little parameter tuning. A similar model was also employed by Shi et al. in [SUEY+17b] which was one of the best entries at the Fourth Dialog State Tracking Challenge. Following their success, we also use a simple design for our CNN to solve the given tasks. Our basic model consists of a convolutional layer with multiple filters and feature maps, followed by a max-pooling layer. Results of the pooling layer are forwarded to the fully connected output layer which utilizes the softmax function to generate the final output of the network. An overview of the architecture can be found in figure 4.1.

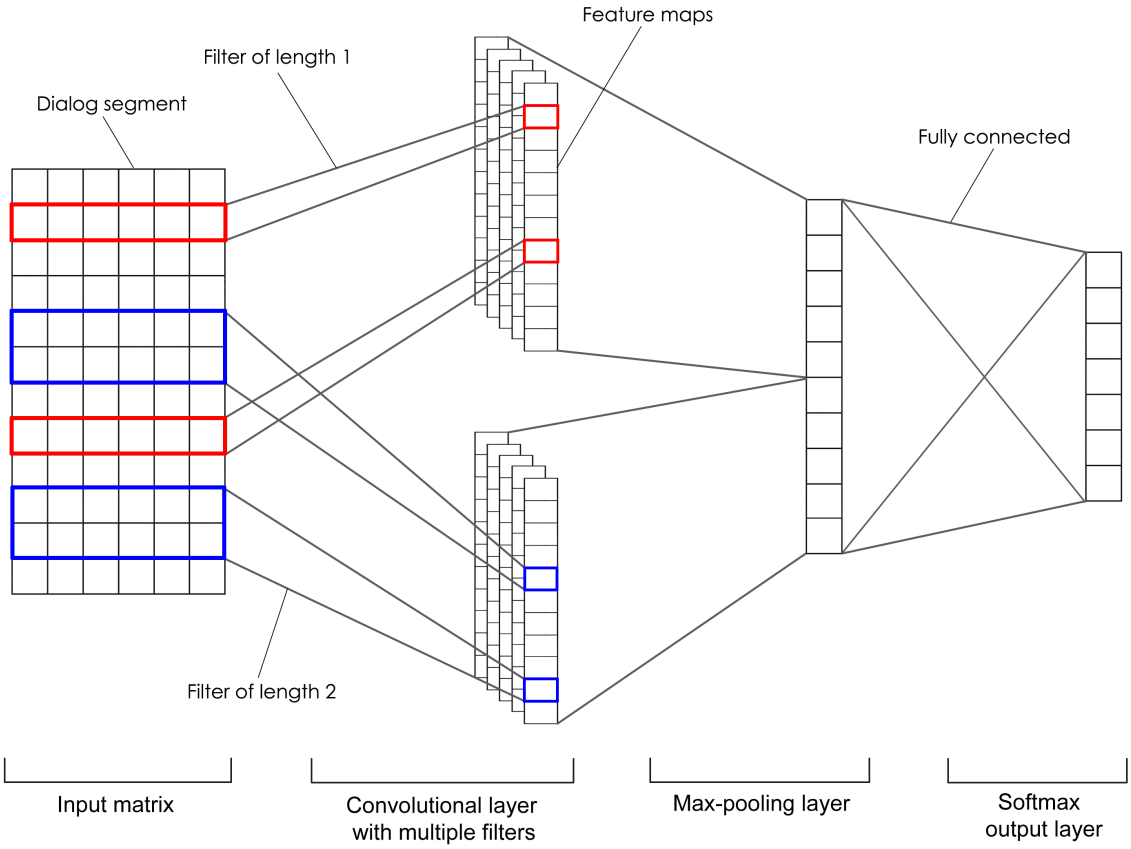

Figure 4.1: Architecture of the basic CNN model. In this example, the model has 2 different filter lengths and 5 feature maps for each of the filter lengths.

4.3.1.1. Model Input

Neural networks perform arithmetic operations on the given data and thus require the input to be numerical values. Therefore a dialog segment consisting of raw words has to be converted to a mathematical representation beforehand. A vector representation of words is desired and can be gained in many different ways. There are commonly used algorithms which train neural networks to produce word vectors in a continuous space. In addition to the word itself, there are also other relevant information that can be added to the vector of one word.

For our CNN model, a vector representation of each word in the dialog segment is obtained by the following combination of feature vectors

$$\mathbf{x} = \mathbf{f}_{word} \oplus \mathbf{f}_{tag} \oplus \mathbf{f}_{speaker}$$

where $\oplus$ denotes the concatenation of two vectors. The first feature vector $\mathbf{f}_{word}$ is a S-dimensional vector in a continuous space representing the given word. It is obtained using a trained `word2vec` model as described in 2.3 where its optimal dimensionality S is determined through further experiments. In order to use the information of the ontology, which is the list of all possible dialog segment labels, the vector $\mathbf{f}_{tag}$ is added to the overall word vector. This tag vector sets the word with the topic label it belongs to, if the substring matches with an entry in the ontology. The tagged topic is then indicated using the 1-of-n encoding scheme which sets the vector element at its index in the ontology to 1, else 0. Since there are 23 topic labels (see section 5.1 for details), this tag vector also has a dimension of $T = 23$. The word vector completes with a 2-dimensional speaker vector $\mathbf{f}_{speaker}$ which indicates the speaker of the utterance this word belongs to. These speaker vectors are given by [1, 0] and [0, 1] for the first and second speaker, respectively. To sum it up, each word in an utterance is represented by a K-dimensional continuous vector where K is equal to $S + T + 2$ and together they form the dialog segment matrices.

A dialog segment matrix consists of all words in the utterances in their occurring order where speaker indicators and punctuation marks are omitted. Each row of this matrix corresponds to the K-dimensional feature vector $\mathbf{x}_{ij} \in \mathbb{R}^K$ of the j-th word in the i-th utterance. Additionally, a zero vector is inserted between every two adjacent utterances. For a dialog segment with n utterances, which are of length $l_1, l_2, ..., l_n$, the dialog segment matrix $\mathbf{M}$ is given by

$$\mathbf{M} = \Big[\mathbf{x}_{11}, \cdots, \mathbf{x}_{1l_1}, \mathbf{0}, \mathbf{x}_{21}, \cdots, \mathbf{x}_{2l_2}, \cdots, \mathbf{0}, \mathbf{x}_{n1}, \cdots, \mathbf{x}_{nl_n}\Big]^T.$$

When we set $N = \sum_i l_i + n - 1$, the total size of this dialog segment matrix can be expressed with $N \times K$ where N is a varying number depending on the lengths of the utterances. To simplify the calculations, the input matrices are made equal-sized by padding with zero row vectors when neccessary.

4.3.1.2. Convolutional and Pooling Layer

As the first step, the input dialog matrices are processed by the convolutional layer. It consists of several sets of filter weights which differ in their lengths. For example, a unigram filter is a filter of length 1 which operates on a single row of the input matrix, where a bigram filter has a length of 2 and works on 2 word vectors at the same time (depicted in figure 4.1 as red and blue, respectively). Each of the filters performs convolution on whole word vectors of length K, hence the total size of a filter with length L is given by $L \times K$.

Through the convolution operation with the filters, feature maps are generated out of the inputs. Let $\mathbf{W}_c \in \mathbb{R}^{L \times K}$ denote a filter weight matrix of length L and with the definition of the convolution operator $*$ from section 2.1.2, the associated feature map $\mathbf{c}$ is calculated by

$$\mathbf{c} = f(\mathbf{X} * \mathbf{W}_c + \mathbf{b}_c).$$

Here, $\mathbf{b}_c$ is a bias vector that is usually added after the convolution operation and thus has the same length as the convolution result which is $N - L + 1$ (N is the length

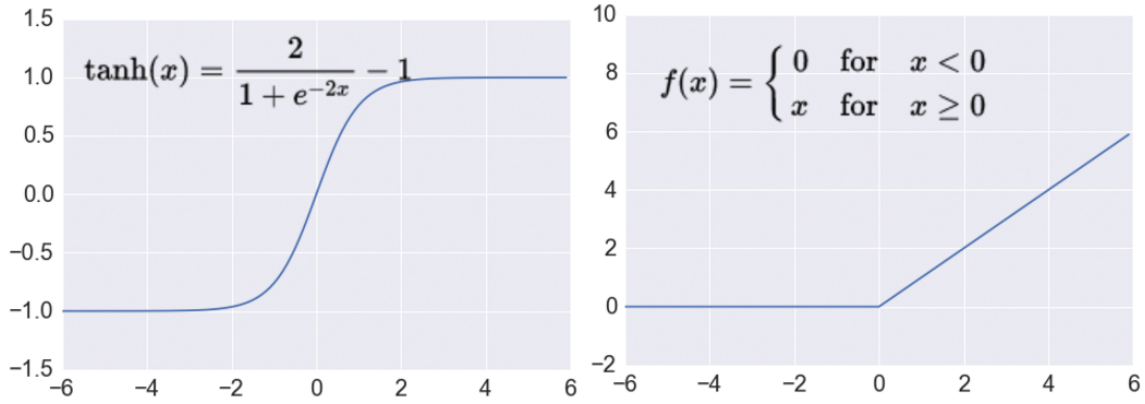

Figure 4.2: Graphs of the non-linear activation functions tanh (left) and ReLU (right). Source: [5]

of the dialog segment matrix as defined in the previous section). The function f acts like an activation function that ensures the non-linearity of the model. Commonly used non-linear functions include the hyperbolic tangent tanh and the rectified linear unit (ReLU) which is defined as

$$f(x) = \max(0, x).$$

ReLU as an activation function was first introduced in [HSMD+00] and used for convolutional networks by Glorot et al. in [GlBB11]. Figure 4.2 displays the graphs of both mentioned funtions. While tanh is traditionally used in many types of neural networks, it has been shown that convolutional neural networks can sometimes be trained more efficiently using ReLU.

After applying the filter $\mathbf{W}_c$ to the input matrix and calculating the activation, a max-pooling operation is applied over the entire feature map $\mathbf{c} \in \mathbb{R}^{N-L+1}$ which takes its maximum value

$$\hat{c} = \max(\mathbf{c})$$

as the feature corresponding to this particular filter. Additional to reducing the number of model parameters, the max-pooling technique is also able to capture the most important feature for each feature map which is the one with the highest value. This pooling scheme also reduces the influence of the dialog segment length. Shorter segments can be treated the same way as longer ones, since the important aspect is not the amount of words but their information content.

As described above, applying one filter extracts one single feature out of the input matrix. By using many sets of filters, multiple features can be obtained. It is also beneficial to experiment with different filter sizes, although a huge number of filters increases the complexity of the network model as well. But since one filter is shared across the entire input, there are less parameters to optimize compared to densely connected neural networks. After applying all the filters, the pooled features are

[5] http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/

concatenated to form a single vector, such that for a set of n filters of varying lengths, the final output $\hat{\mathbf{c}}$ of the convolutional and pooling layer is

$$\hat{\mathbf{c}} = [\hat{c}_1, ..., \hat{c}_n]^T$$

where $\hat{c}_1, ..., \hat{c}_n$ are the features extracted from each of the filters. This concatenated output vector is then passed to the fully-connected output layer for a further processing step.

4.3.1.3. Output Layer

Goal of the output layer is to generate a meaningful result out of the network computations. This layer consists of the same number of neurons as there are topic labels to classify. It is fully connected to the convolutional and pooling layer which means that every neuron in this layer is connected to each output unit of the previous layer. Let $\hat{\mathbf{c}}$ again denote the concatenated features from the previous layer, the output layer computes the following

$$\mathbf{o} = \hat{\mathbf{c}} \cdot \mathbf{W}_o + \mathbf{b}_o$$

where $\mathbf{W}_o$ and $\mathbf{b}_o$ are the weight matrix and bias vector of this layer, respectively. The result $\mathbf{o}$ is a vector of the same dimension as the number of topic classes and thus contains information for each of the classes.

It is desirable to gain a result that is easy to interpret. Therefore, we additionally apply the softmax function σ which is elementwisely defined as

$$\sigma(\mathbf{o})_j = \frac{e^{o_j}}{\sum_{t=1}^{T} e^{o_t}} \quad \text{for } j = 1, ..., T$$

for a T-dimensional vector $\mathbf{o}$ and e as the natural exponential function. It has the property to convert vector elements of arbitrary values to values between 0 and 1 which sum up to 1. This implies that the output of the softmax function can be directly used as a probability distribution over the topic classes. Further information on the softmax function can be found in [Bish06]. To determine the final classification result, we can simply take the index of the maximum argument from the output vector.

4.3.2. Multilayer Network

As described in the previous section, the basic model consists of one convolutional and pooling layer aside from the output layer. Since this design is comparatively simple, it would be interesting to experiment with architectures that are slightly more complex. Therefore we extend our first model by adding more depth in form of one or more hidden layers. Figure 4.3 displays the structure of this second CNN architecture. Additional to the already described layers, the network consists of two hidden layers which are fully connected to their neighboring layers.

Hidden layers are layers that reside between the input and output layer. Unlike input neurons that take information from the external world or output neurons

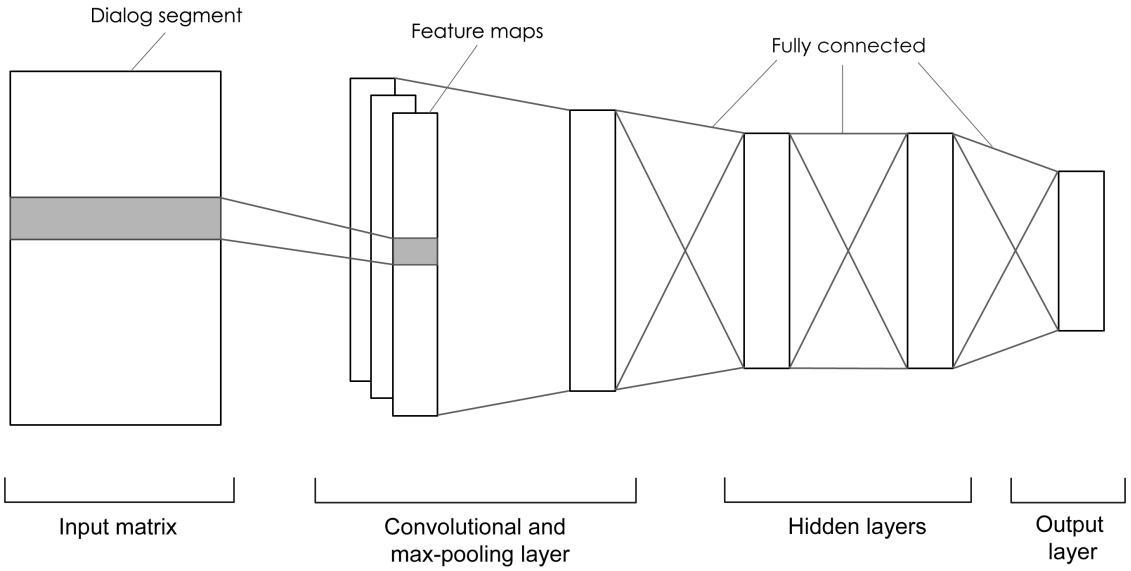

Figure 4.3: Architecture of the multilayer CNN model with two fully connected hidden layers.

whose values can be observed from the outside, hidden neurons are not connected to the environment and only communicate with other neurons. Hidden units act as feature detectors for some types of inputs. By combining these features, the output units can sometimes perform more powerful classifications than they can without the hidden units. As mentioned above, the hidden units are connected to every output unit of the previous layer. Their activity depends on the neuron inputs and the weights on the connections between the previous layer and the hidden units. The output $\mathbf{h}_i$ of the i-th hidden layer can be computed with

$$\mathbf{h}_i = f(\mathbf{h}_{i-1} \cdot \mathbf{W}_{h_i} + \mathbf{b}_{h_i})$$

where $\mathbf{W}_{h_i}$ is the weight matrix and $\mathbf{b}_{h_i}$ the bias vector of this hidden layer. The expression $\mathbf{h}_{i-1}$ denotes the output of the previous hidden layer where $\mathbf{h}_0$ is the output of the max-pooling layer. Similar to the convolutional layer, a non-linear activation function f is used to determine the final activity of the hidden units. Commonly used hidden activations are the hyperbolic tangent or the sigmoid function. After propagating the inputs through the hidden layers, the outputs produced by the last hidden layer are then passed to the units of the output layer.

The example in figure 4.3 displays a network with two hidden layers. In fact, the number of hidden layers should be kept small since the dense connections increase the network complexity by a great amount of parameters. Hence, one or at most two hidden layers are feasible choices. Also the number of units in a hidden layer should not be very large. A smaller hidden layer not only increases the rate at which the network can be trained, it can also help to improve the overall performance. With too many hidden units, the network tends to memorize the correct label for each example in the training set instead of learning a general solution. By limiting the number and size of the hidden layers, the network is forced to develop more

generalized feature detectors and will probably show better performance for unseen inputs.

4.3.3. Word Embedding Network

In section 4.3.1.1 it was stated that for the basic model, words of the input dialog segments are converted into vector representations using pre-trained `word2vec` models before they are processed by the network. Neural network classifiers are also able to learn continuous word representations themselves, so it is possible to directly use the raw input dialog data with minimum preprocessing. This can be achieved by employing a projection layer which receives the input and generates the dialog matrices required by the subsequent convolutional layer. An overview of the word embedding architecture can be found in figure 4.4.

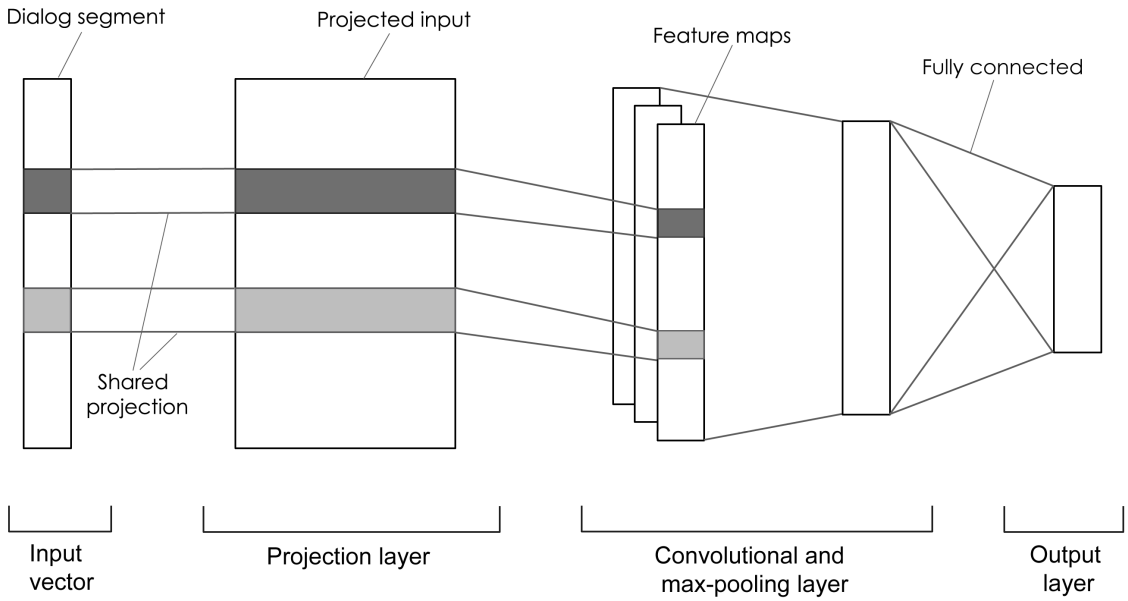

Figure 4.4: Architecture of the word embedding CNN model with an additional projection layer.

The projection layer maps all words of an input dialog segment to a continuous vector space. Since it requires the input words to be numeric values, the 1-of-n encoding scheme is employed. Each word is then represented by a vector of binary valued elements of length equal to the size of the vocabulary which is a list of all words occurring in the dataset. For a word with the index i in the vocabulary, only the i-th element of the vector is set to 1 and all other values are set to 0. Since the words will be represented by lots of zeros in this way, a more compact method is to represent the words only by their indices. Hence the input dialog segment is defined by a vector consisting of word indices where the index 0 is inserted between two adjacent utterances.

The number of units in the projection layer is specified by the desired size of the continuous space where the word representations are taken from. Each neuron in the projection layer is represented by a number of weights equal to the size of the vocabulary. These weights form together the projection matrix which is shared by

all inputs such that the same words in a dialog segment also receive the same projected vector. Using the standard 1-of-n encoding, projection can be performed by multiplying the projection matrix with the input. But since the encoded vectors are reduced to word indices, the output for a particular word with index i is simply the i-th row of the projection matrix. Like parameters of other layers, the projection matrix is also learned during training, so the network can optimize the matrix weights to find word representations that produce the desired output. Like for the basic model, the word embedding vectors are extended by tag and speaker vectors $\mathbf{f}_{tag}$ and $\mathbf{f}_{speaker}$ which are defined in the same way as previous. The concatenated feature vectors for each word form the dialog segment matrix which is used as the input to the convolutional layer.

5. Evaluation

We train several neural networks for each of the architectures described in the previous section 4.3 on the labeled Talkbank corpus obtained in section 4.2. These networks are trained on three different tasks which are tracking, next topic prediction and topic change prediction. Various experiments are conducted to analyze the impact of different model and training parameters on the network performance and to find optimal values for each of them. The regularization techniques from section 2.1.4 are also extensively evaluated to improve the generalization ability of the model. While performance is an important factor, it is also necessary to consider the time needed to train the networks. Since training large neural models usually requires a lot of time, it is important to compromise between the model's complexity and the effort to train it, as well as to stop training early if no more significant improvement is observed. For the vector space word model described in section 2.3, we also train various models with different sets of parameters and test the performance on sample tasks. The following sections provide more details on the labeled dialog corpus and summarize the most important results obtained in the course of this work.

5.1. Data Analysis

Overall, the resulting training dataset is taken from 41 conversations and sums up to 10,284 utterances and 122,541 words. The size of the vocabulary used in these conversations amounts to 5,197 unique words. A list of all labeled files from the BilingBank corpora can be found in table A.1 in the appendix. These dialogs are split into 1,714 dialog segments which are labeled using one of 23 distinct topic labels found during the labeling process described in 4.2. Figure 5.1 shows how the assigned topics are distributed over the entire labeled dataset. Most frequent topics are *Acquaintance* with 161 occurrences, followed by *Work*, *Health* and *MovieAndTV* with over 120 occurrences each. While *Acquaintance*, *Work* and *Health* are topics that appear in many conversations, *MovieAndTV* is a topic that is only found in a few dialogs, but is talked about for a very long time. Least frequent topics include *Car*, *Politics* and *Location* with only about 30 examples each. All topic labels are

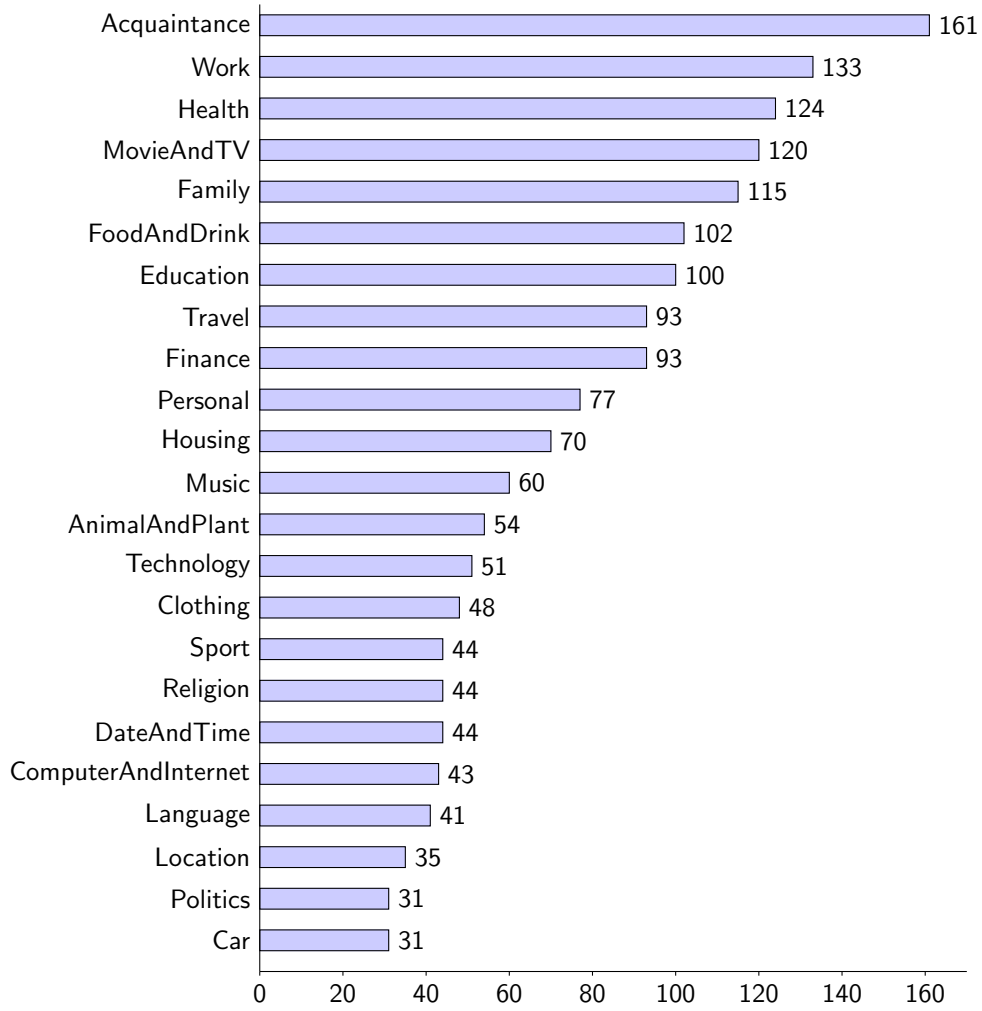

Figure 5.1: Distribution of the 23 topic labels over all labeled dialog segments.

further summarized in table B.1 in the appendix along with examples of subtopics covered by each of the labels.

Usually, the whole dataset should be divided into three subsets, for training, validation and testing, in order to strengthen the network against overfitting. But due to the limited amount of labeled data and the complexity of the tasks, there are not enough data for some to be used for validation and testing purposes only. Therefore, the k-fold cross-validation scheme is applied. The idea is to break the training data into k subsets and train the same number of neural networks on these subsets. Each of the networks uses a different subset for testing, then randomly chooses one of the remaining subsets for validation and the last $k-2$ subsets are utilized for training. After training the k models, their performances are averaged as an estimate of the overall network performance. The advantage of this method is that all examples are used for both training and testing, and each example is used for testing exactly once.

In our case, the training dataset is divided into $k = 10$ subsets, which are hence called the test cases. These test cases should be similar in their sizes and ideally also have comparable distributions over the topic labels. But since examples of one dialog need to stay together to keep the conversation flow, it is unfortunately not possible to achieve similar label distributions for all test cases. Moreover, there are

many topics that are insufficiently represented in some test cases and a few topics that do not appear there at all. This creates a significant difference between these test cases, so deviations in network performance depending on the test case used can be expected. Details on the exact number of each topic label contained in the test cases can be found in table 5.1 below. Additionally, the number of topic change occurrences in each test case can be found in the last row. Table A.2 in the appendix further lists for every test case which conversation files belong to it.

Topic label	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
Acquaintance	9	16	16	0	18	10	1	17	40	34
AnimalAndPlant	0	17	4	4	0	0	19	0	4	6
Car	0	6	0	8	0	9	5	3	0	0
Clothing	5	1	12	0	9	2	4	9	6	0
ComputerAndInternet	0	0	0	0	0	0	11	0	13	19
DateAndTime	5	0	19	0	6	5	7	0	2	0
Education	4	15	23	7	0	2	10	6	22	11
Family	11	10	7	13	14	19	21	12	3	5
Finance	22	0	10	6	6	10	7	20	11	1
FoodAndDrink	0	16	17	4	12	23	0	13	5	12
Health	9	36	8	10	6	32	0	18	5	0
Housing	16	2	2	11	10	0	3	8	9	9
Language	21	0	0	3	7	0	5	0	5	0
Location	3	0	0	18	0	2	10	2	0	0
MovieAndTV	2	3	18	28	33	5	0	0	3	28
Music	0	0	5	17	0	5	18	0	0	15
Personal	0	3	32	8	11	1	0	10	6	6
Politics	0	0	5	0	0	4	0	7	15	0
Religion	10	9	0	0	0	9	16	0	0	0
Sport	10	4	5	0	2	6	4	4	9	0
Technology	5	3	0	2	7	3	4	0	16	11
Travel	9	14	13	11	12	9	0	6	15	4
Work	21	7	16	11	7	5	17	25	22	2
$\sum$	162	162	212	161	160	161	162	160	211	163
Topic change	27	36	47	27	40	38	39	35	41	30

Table 5.1: Number of examples of each topic label for every test case T1 to T10. The last two rows show the overall number of examples and the number of topic changes, respectively. While most test cases are about the same size, it is difficult to achieve similar topic distributions among them.

5.2. Evaluation Metrics

The performance of a network is evaluated by comparing its calculated outputs with the reference labels. Two kinds of evaluation metrics are employed to measure the accuracy of the models. One of them is error rate, which is defined as the percentage of model outputs that do not match with the reference labels for a given dataset and is mathematically expressed by

$$\text{error rate} = \frac{\text{number of wrong outputs}}{\text{total number of outputs}} \cdot 100.$$

Error rate is a simple way to evaluate the accuracy of neural networks, but it typically does not provide enough information on whether the model is really a good classifier or not. This particularly happens when the dataset is greatly unbalanced, which means that the number of instances for each class is very different from each other. For example on a dataset with two classes where 80% of the instances belong to the first class and 20% to the second, a binary classifier could achieve an error rate of 20% by always returning the first class. A relatively low error rate of 20% might be interpreted as a good performance, but the model is not very accurate since it will classify every instance of the second class wrongly which can be serious mistakes in some cases.

A better way of measuring a model's accuracy is the F-measure or F-score and it is computed by combining two other metrics which are the precision and the recall. For classification tasks with multiple class labels, they can be calculated using the confusion matrix. Table 5.2 shows a simple confusion matrix which summarizes the results of a classification task involving 3 classes. Each of the rows represent a model output class where the columns contain the reference classes. Therefore an entry in the ith row and jth column is the number of results that are predicted as class i by the model but actually belong to class j. Ideally, the confusion matrix is a diagonal matrix consisting of non-zero entries only when $i = j$ or at least have negligibly small values for all the other entries.

	Class 1	Class 2	Class 3
Class 1	88	2	2
Class 2	4	24	0
Class 3	2	0	17

Table 5.2: Example of a simple confusion matrix with 3 classes and the corresponding classification results.

Precision and recall are calculated for each of the classes separately. While precision for a class i is defined as the fraction of correct outputs among all class i outputs made by the model, recall reflects the fraction of correctly predicted class i outputs over the total amount of class i reference labels in the dataset. Given a confusion matrix $\mathbf{C}$ with N classes, precision and recall for class i can be calculated by

$$\text{precision}(i) = \frac{\mathbf{C}_{ii}}{\sum_{j=1,\dots,N} \mathbf{C}_{ij}}, \quad \text{recall}(i) = \frac{\mathbf{C}_{ii}}{\sum_{j=1,\dots,N} \mathbf{C}_{ji}}.$$

A high precision for a class means that more correct results than incorrect ones were returned and the model is more exact, whereas a high recall means that most instances of that class were found correctly and the model is more complete. Trying to increase only precision or recall would usually reduce the other, hence it is required to balance both values and combine them into a single measure, such as the F-score. The traditional F-score is defined as the harmonic mean of precision and recall. For class i it becomes

$$\text{F-score}(i) = 2 \cdot \frac{\text{precision}(i) \cdot \text{recall}(i)}{\text{precision}(i) + \text{recall}(i)}$$

and the overall score can be obtained by averaging the scores for each of the classes. Like precision and recall, F-score reaches its best value at 1 and worst at 0. While training on error rate only minimizes the overall model error, using F-score optimizes the model performance on the two different aspects of model accuracy and is generally a more significant measurement for tasks with high class imbalance. Hence instead of error rate, we use F-score to train our network models and also in most parts of the evaluation.

5.3. Training Speed

The time required to train our CNN models strongly depends on the hyperparameter settings. A small network can be trained in less than 10 minutes on a NVIDIA Tesla M2075 GPU where for a larger regularized model, it can take more than 100 minutes for it to converge. But actually, a near convergence performance is usually already achieved after a few epochs.

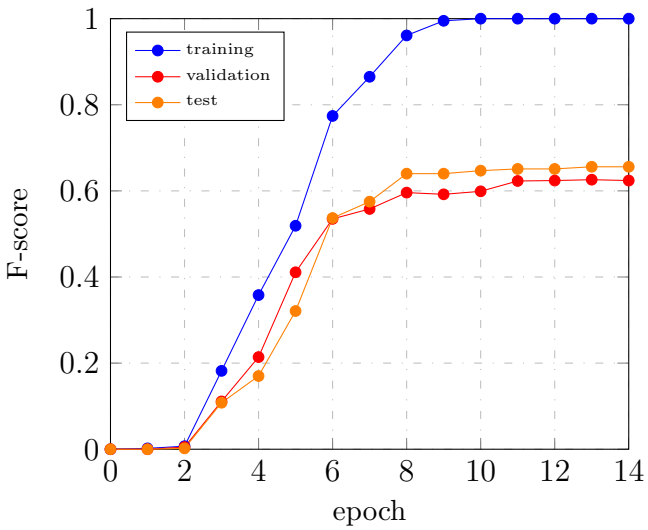

Figure 5.2: Model performance during training on the training, validation and test dataset.

Figure 5.2 shows the performance of the first 15 training epochs on the training, validation and test datasets. The model is trained on the tracking task, with a learning rate of 0.1, filters of size 1 and 2, and 200 filters each. For the other two tasks, a similar behavior can be observed. As shown in the diagram, the maximum performance on the training set is already hit after 10 epochs and the model starts to converge on the validation and test sets at the same time as well. But training usually lasts for many more epochs while going back and forth before it finally converges. In this case, it ends after the maximum number of 200 epochs is reached and is able to obtain a test performance of 0.71. Since this is a significant improvement compared to the score of 0.65 after 15 epochs, it is usually still beneficial to continue training despite the additional effort.

5.4. Word Model

As described in section 2.3, the word model to convert raw words into continuous vector representations can be trained using gensim's `word2vec` tool. These models are trained using a very large Wikipedia corpus which can be found on the same page as the Wikipedia text8 corpus. It consists of 1 GB of English Wikipedia articles and has got a vocabulary of over 1.4 million words. Since these texts are directly taken from Wikipedia, they contain special text parts which are not contents of the actual articles like XML tags and links. Hence the text corpus needs to be cleaned up first by removing all the unnecessary parts. The remaining clean text files can then be passed to gensim's text8 iteration method to generate sentences which can be used to train the word model.

Since `word2vec` training is an unsupervised task, there is no good way to objectively evaluate the result. Google's research group has released a testing set of over 23,000 syntactic and semantic test examples with each consisting of four words. In all examples, the words follow the rule that the relationship between the first two words is the same as between the last two words, e.g. "Athens Greece Berlin Germany" which describes a country and its capital. The testing set is divided into 18 categories where four of them consist of examples with phrases, which are compositions of two or more words. An overview of all test categories can be found in table C.1 in the appendix. Gensim supports this evaluation set in exactly the same format as the original tool and it returns the number of correct answers for each class separately as well as an overall model accuracy. Although better performance on this test set does not necessarily mean that the word model will also work better on our tasks, but it is still worth trying to optimize the model and compare the performance in combination with our neural network models.

Gensim's implementation provides a large range of parameters and we experiment with various parameter values to obtain a model with a greatly improved performance. Using only the default configuration, the trained model evaluates 6,057 of 11,847 test examples correctly which relates to an accuracy of roughly 51.1%. While some of the parameters achieve the best results at their default values, there are a few other parameters that can be tuned to increase the model's accuracy. One of them is the size of the word vectors, the dimension of the vector space where the word representations lie. Since the default value at 100 is relatively small, we train models with increased sizes without changing the other parameter values. The experimental results can be found in figure 5.3.

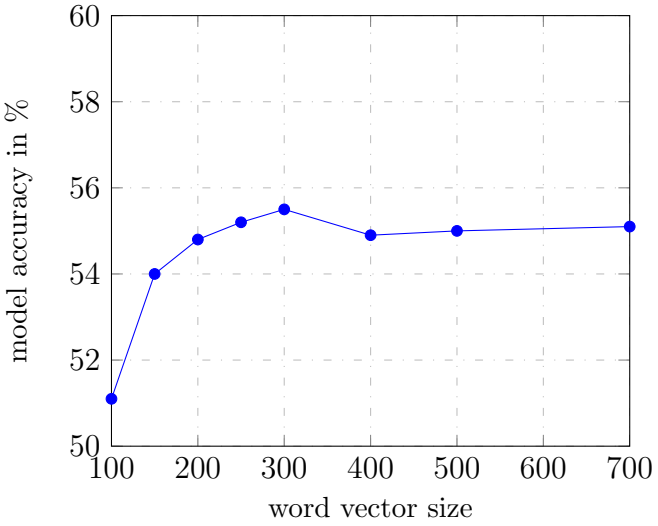

Figure 5.3: Overall accuracy of word models trained with different vector sizes and other parameters at default values.

While the performance can be improved by increasing the vector size, it apparently converges after reaching a vector size of 300 and peaks at an accuracy of 55.5%. Training each of the sizes takes about the same time on a NVIDIA Tesla K20m GPU, which is roughly 50 minutes. Although increasing the dimensionality does not require more time to train, it might introduce new complexity to the CNN and therefore a simpler word model should be preferred.

Another significant factor is the initial learning rate which drops linearly as training progresses. Like for the word size, an improvement can be observed by changing the given default value of 0.025 as shown in figure 5.4. Also here, a convergence occurs after reaching a model accuracy of around 55% at a learning rate of 0.05. When increasing the learning rate beyond 0.13, the performance rapidly falls down to under 1%.

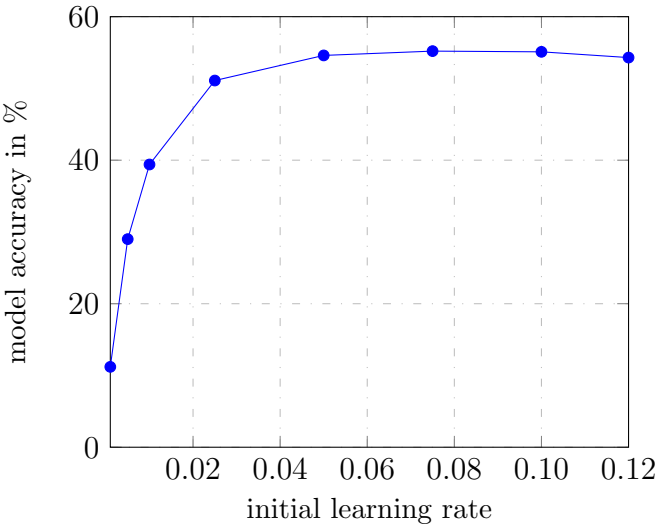

Figure 5.4: Overall accuracy of word models trained with different initial learning rates and other parameters at default values.

Hence the best model performance can be achieved by choosing a learning rate between 0.05 and 0.1. Similar to the experiments on word size and learning rate, several other parameters are evaluated as well and the results can be found in figure C.1 in the appendix. Since by simply choosing the best value for each of the parameters does not yield a very good performance, further experiments are necessary to find the optimal combination of parameters. Our final best model is trained with, among others, a vector size of 200, a learning rate of 0.05 and 15 iterations over the corpus. Training the model takes about 2 hours 44 minutes and it achieves an overall accuracy of 71.5% on the test data. A detailed result for each test category can also be found in table C.1.

Since a better word vector model does not automatically improve the overall performance on the given tasks, further evaluations in combination with the CNNs are required. Therefore, four word models with quite different accuracies are tested on each of the three tasks and the results are summarized in figure 5.5.

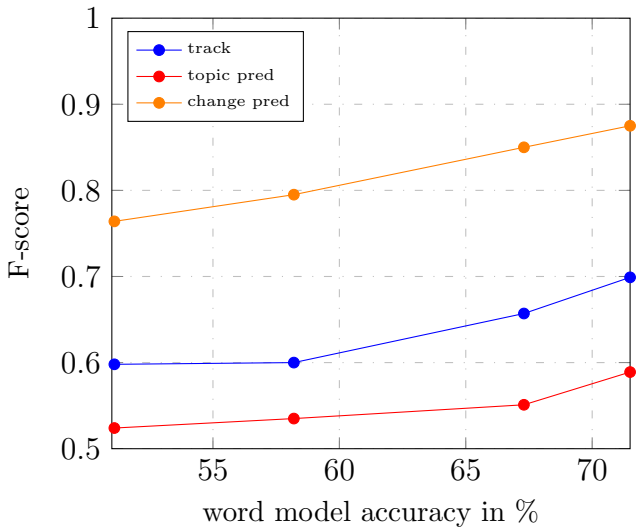

Figure 5.5: Results of experiments on the correlation between word model accuracy and overall performance of the CNN for each of the three given tasks.

The CNNs are all trained on test case 1 and they also share the same set of parameters. The basic model described in 4.3.1 is used with a convolutional layer consisting of filters of size 1 and 2, and 200 filters each. Learning rate is set to 0.1 for training and no regularization is applied. As seen in the experimental results above, a significant performance improvement can be observed for each of the three tasks. For the current topic tracking task, the F-score increases from 0.598 with the word model trained with default parameters to 0.699 with the best word model which is an excellent improvement of over 0.1. An even greater F-score increase of about 0.12 from 0.764 to 0.875 is achieved for the topic change prediction task. And finally, the performance for the next topic prediction task increases from 0.524 to 0.589, a slightly lower improvement of 0.065 compared to the other two tasks.

5.5. Hyperparameters

Our CNN models are trained with several hyperparameters which are set to fixed values and are not changed during the entire training process. To evaluate each

of the hyperparameters in order to find optimal values, models are trained with varying values of one parameter while keeping the remaining ones unchanged. Since this optimization step is very time consuming, experiments are conducted with the basic model only because a similar behavior can be expected for the other two architectures. It is also not possible to test the various hyperparameter values on all 10 test cases, hence we limit our experiments to test case 1 and try to find the best parameter combination given these restrictions. For most of the experiments, models are trained for all three given tasks and the performances are compared. The following sections provide the results for the most interesting parameters, which are filter size, number of filters and the learning rate. A brief summary of results on other parameters can also be found below.

5.5.1. Filter Size

The first experiments are conducted to investigate the impact of different filter sizes. Therefore, we apply 300 filters of the same size and compare the performance on each of the three tasks. All models are trained with a learning rate of 0.1 and without any form of regularization. The results are displayed in figure 5.6. Note that filter size refers to the height of the filters, i.e. the number of words to be processed at once. The width of the filters is equal to the length of the word vectors, so the actual size of the filters is filter size times word vector length.

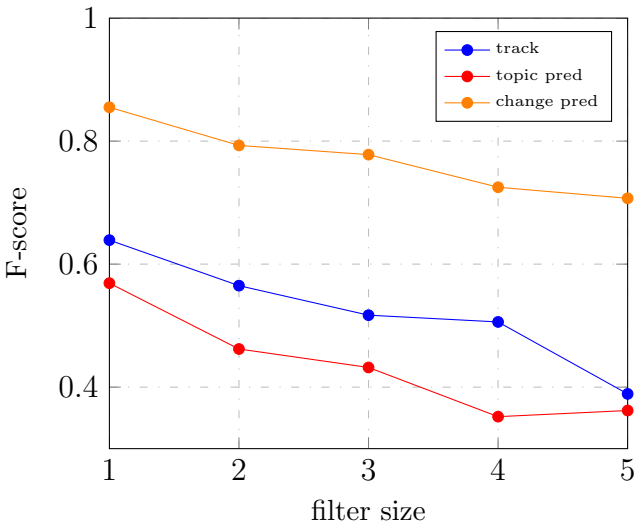

Figure 5.6: Results of experiments on different filter sizes for the convolutional layer where 300 filters are used for each of the sizes.

As shown in the results, unigram filters are able to perform best and bigram filters are close behind. Filters that span over four or more words loose a significant amount of performance, especially for the tracking and topic prediction tasks. Although there are local correlations in natural languages like in images, our CNNs do not seem to benefit from these correlations. A possible explanation is the sparsity of the training data, since longer word sequences are most likely rarely seen during training and the network is unable to properly learn from them. Since the experiments only employ filters of one size, it would also be interesting to combine filters of different sizes which is further investigated in the following section.

5.5.2. Number of Filters

Similar to the experiments on the size of the filters, we train models with various numbers of unigram and bigram filters. For example if the number of filters is set to 100, this means that 100 filters of each filter size are used and hence results in overall 200 filters. A comparison of the performances can be found in figure 5.7.

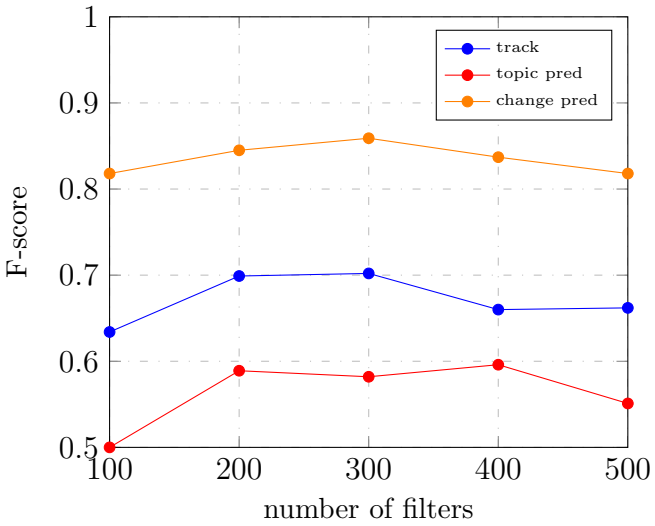

Figure 5.7: Results of experiments on different numbers of filters for the convolutional layer where filters of size 1 and 2 are used.

Although the F-score can be improved by increasing the number of filters of each size from 100 to 300 (400 for the topic prediction task), it worsens again after a further increase. Despite the fact that training with 1000 filters takes 50% more time than 600 filters, it does not achieve the desired improvement. Like for the filter size, the reason for this behavior can be the sparsity of training data which is not enough to train large networks and causes them to overfit. Further experiments on different combinations of filter sizes and number of filters are conducted and the results can be found in table 5.3.

Filter sizes	Number of filters	Track	Topic pred	Change pred
1, 2, 3	200	0.66	0.60	0.83
1, 2, 3	300	**0.70**	**0.61**	0.82
1, 2, 3	400	0.66	**0.61**	**0.84**
1, 2, 3, 4	200	0.68	0.57	**0.84**
1, 2, 3, 4	300	0.66	0.55	0.77

Table 5.3: Performance of different combinations of filter sizes and number of filters. The last three columns contain the F-scores for the tracking, next topic prediction and topic change prediction tasks, respectively.

For the tracking and next topic prediction tasks, the performance can be improved by adding trigram filters in addition to the unigram and bigram filters. This is not the case for the topic change prediction task because training for this task is probably more vulnerable to overfitting since there are considerably less ”change” than ”no change” examples.

5.5.3. Learning Rate

We also evaluate different values for the learning rate, a parameter defined in the previous section 2.1.3 which influences both the quality and the speed of the training algorithm. Usually, the learning rate should be neither too small nor too large in order to achieve good results in relatively short training periods. Figure 5.8 gives a summary of the experimental performances. All models employ filters of size 1, 2 and 3, and use 300 filters of each size.

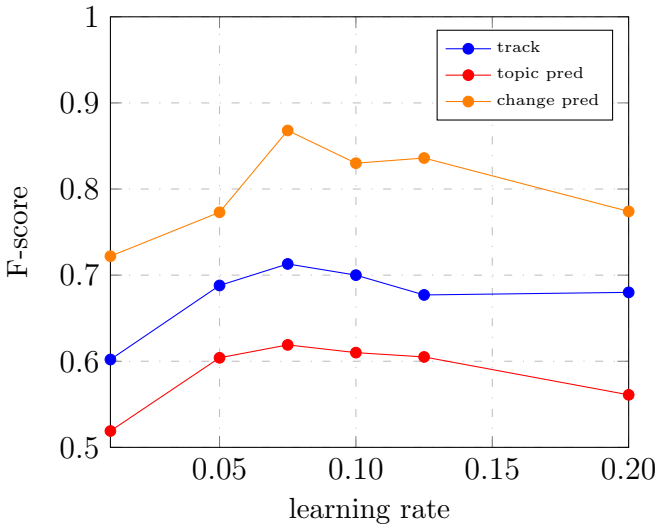

Figure 5.8: Results of experiments on training CNNs with different learning rates.

A similar behavior can be observed for all three tasks with each of them achieving the best performance at a learning rate of 0.075. Since in most cases training continues until the maximum number of epochs is reached, the training time is about the same for each of the learning rates and lies between 30 and 40 minutes.

5.5.4. Other Parameters

Besides the three above mentioned hyperparameters, there are a few other parameters that can be fine tuned. One of them is the size of the mini batches which defines how many examples are processed by the model at once. Therefore the training data is divided into batches of the specified size where only full batches can be used and the remaining ones have to be omitted. The stochastic gradient descend algorithm is altered to iterate over these batches instead of single examples and uses the batch average likelihoods for calculating the gradients. Hence the batch size is a parameter that can have influence on the overall model performance. In order to loose as little examples as possible, only small batch sizes are tried out. The results show that a size of 50 examples per batch performs best which leads to 26 or 27 mini batches for training depending on the test case used.

Another parameter that can have an impact on the model's performance is the length of the dialog history. It specifies the number of previous dialog segments which are used as network input in addition to the current dialog segment. All other hyperparameter experiments are conducted on a history length of 0, i.e. without the use of history. By including a history of one dialog segment in the input, the

model is able to achieve an F-score improvement of 0.02 on the tracking task, but no performance gains on the other tasks. Increasing the history length even more unfortunately only worsens the performance on all three tasks. Considering the additional time needed to train the networks, which amounts to more than twice the usual time, and the relatively low performance gain, we hence omit the use of the history.

As described in 4.3.1.2, there are two non-linear functions that are often used as activation function in CNNs. We evaluate both functions on all layers that implement an activation and the results demonstrate no significant differences between the two schemes. Only for the tracking task, the ReLU function performs slightly worse than tanh, although it is commonly better for convolutional layers. Since this is not the case for our models and tasks, we use tanh as the non-linear activation function for all layers. Detailed evaluation results for all parameters described in this section can be found in table D.1 in the appendix.

5.6. Regularizations

After finding good values for each of the hyperparameters, we implement the regularization schemes described in section 2.1.4. For L1 and L2 regularization weights, values between 10^{-6} and 10^{-3} are tested. The results can be found in figure 5.9 where the models are trained with a learning rate of 0.075 and 300 filters of size 1, 2 and 3.

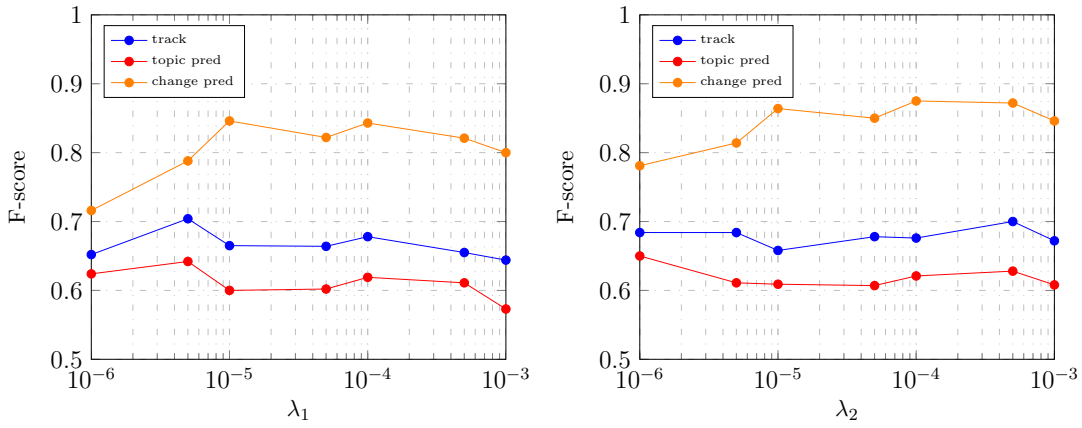

Figure 5.9: Results of experiments on different L1 (left) and L2 (right) regularization weights λ_1 and λ_2.

Interestingly, the tasks tracking and next topic prediction show very similar behaviors for both L1 and L2 weights. But while the best models for the topic prediction task are able to outperform not regularized ones by 0.02 with L1 and 0.03 with L2, regularized models for the tracking task do not perform better than not regularized ones. The L1 regularized models for the topic change prediction task are also not able to outperform not regularized ones, but the best L2 regularized model achieves an improvement of 0.01. Overall, the performance gains from L1 and L2 regularizations unfortunately are not as significant as desired for our CNNs and tasks.

Experiments are also conducted on dropout probabilities for different layers. Table 5.4 provides an overview of some obtained results.

Filter sizes	Number of filters	Dropout prob	Track	Topic pred	Change pred
1, 2, 3	300	0.1, 0.0, 0.0	0.71	0.58	0.83
1, 2, 3	300	0.0, 0.1, 0.0	0.66	0.60	0.83
1, 2, 3	300	0.0, 0.0, 0.1	0.69	0.59	0.85
1, 2, 3, 4	300	0.1, 0.0, 0.0	**0.74**	0.62	0.86
1, 2, 3, 4	400	0.1, 0.0, 0.0	0.67	**0.66**	**0.87**
1, 2, 3, 4	400	0.1, 0.1, 0.0	0.69	0.63	0.86
1, 2, 3, 4	400	0.1, 0.0, 0.1	0.67	0.63	0.83

Table 5.4: Results on dropout probabilities applied to each of the layers. The three numbers in the third column indicate the dropout probability for the convolutional, pooling and output layer, respectively. The last three columns contain the F-scores for the tracking, next topic prediction and topic change prediction tasks.

Dropout is applied to the convolutional, pooling and output layer which is indicated by the three probability values in the third column of table 5.4. At first we test different dropout probabilities on models with 300 filters of size 1, 2 and 3 (row 1 to 3 in the table), and none of them is able to outperform models of the same size with no dropout. Increasing the dropout probability to 0.2 or 0.3 in any layer mostly lowers the F-score. Since the dropout scheme is usually employed for large networks, we further test it on two bigger models (row 4 and 5). These models achieve new best scores for each of the three tasks which improve the former bests by 0.02, 0.04 and 0.01, respectively. Row 6 and 7 contain results of models with two dropout layers, but neither combination is able to outperform the models with only the convolutional layer as a dropout layer.

We also test various combinations of dropout probabilities and L1 and L2 weights, but no further improvement compared to using dropout alone can be observed in most of the cases. Only for the topic change prediction task, the F-score can be improved by 0.01 by adding a very small L2 term in addition to the convolutional layer dropout. While L1 and L2 regularizations do not influence the training time, dropout networks require about 50% more time to train. Although the two regularization schemes are able to slightly improve the performance of the CNNs when used alone, these improvements unfortunately do not add up when combining both approaches. Since dropout applied to the convolutional layer is able to achieve better results regardless of the task, it is hence the preferred regularization method.

5.7. Model Variations

As described in section 4.3.2, a slightly more complex model is also employed in addition to the basic architecture. The new complexity is introduced by adding one or more hidden layers, but because of the sparsity of the data, only one hidden layer is used. Figure 5.10 shows models trained with different hidden layer sizes where the remaining parameters are the same as in the previous section. Hidden layers with 300 or more neurons highly suffer from the large amount of parameters and are hence trained with a dropout probability of 0.5. This enables huge performance improvements ranging from 0.05 to over 0.1 which can be observed across all three tasks. Since the results without dropout are rather poor, figure 5.10 already displays the improved results.

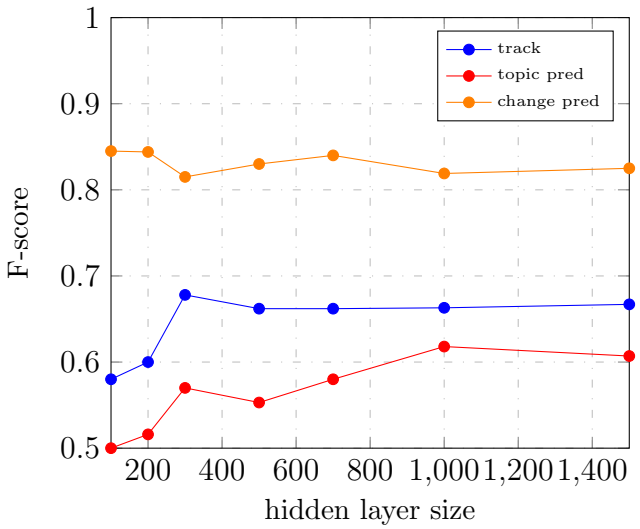

Figure 5.10: Results of experiments on CNNs with one hidden layer of various sizes. Hidden layers of 300 or more units are trained with a dropout probability of 0.5 while smaller hidden layers do not employ dropout.

Similar behaviors can also be found here between the tracking and the next topic prediction tasks. While models for these two tasks perform badly with small hidden layers, for the topic change prediction task, they achieve their maximum with a hidden layer of only 100 units. But despite the excellent improvements through dropout, our best models with the additional hidden layer still do not outperform the best basic models with scores that are close behind. This implies that we probably do not even have enough data to effectively train one additional hidden layer which causes the overall network performance to decrease.

The shortage of training data is even more visible for the word embedding architecture described in section 4.3.3. These models should be able to learn word representations which capture optimal word associations for our data and tasks. Hence, the CNNs should also perform better compared to using word embeddings trained on a different corpus. We evaluate models with various projection layer sizes, but unfortunately they perform poorly for all three tasks. For example the best F-score for the tracking task is 0.34 which is achieved with a projection layer size of 500 and is far below the best score of 0.74 by using pre-trained word vectors. Similarly, for the next topic prediction task, the best result is 0.28 when setting the size of the projection layer to 300. It shows that our corpus consisting of roughly 480 KB text data is simply not enough to train meaningful word embeddings compared to the 1 GB corpus used for `word2vec`. To further investigate this assumption, a `word2vec` model is trained on our much smaller dialog corpus and employed in the basic model. As expected, the results are lower than the ones achieved by the word embedding models with for example a F-score of 0.27 for the tracking task.

5.8. Final Results

A comparison of the best models for each of the three network architectures can be found in table 5.5. All models are evaluated on test case 1 where the multilayer network consists of a hidden layer of size 300 and the word embedding network uses a projection layer of size 500.

Model	Track	Topic pred	Change pred
Basic	**0.749**	**0.642**	**0.826**
Multilayer	0.678	0.570	0.815
Word Embedding	0.346	0.275	0.725

Table 5.5: Comparison of the best models for each of the three architectures. All models are evaluated on test case 1. The F-scores for each of the tasks are given by the last three columns.

Since both the multilayer and word embedding architectures are unable to outperform the basic network, the latter is employed for our final models which are trained on each of the 10 test cases for cross-validation. All 30 models are trained using a learning rate of 0.075, a batch size of 50, no dialog history and tanh as the activation function. The models for the tracking task use 300 filters of size 1, 2, 3 and 4, and a dropout probability of 0.1 in the convolutional layer. A similar configuration is also employed for the next topic prediction task except for the number of filters which is set to 400. Also for the topic change prediction task, 400 filters of each size are used and an L2 regularization weight of 0.00005 is applied additional to dropout. Table 5.6 provides the results achieved by our final models. They are evaluated on each of the 10 test cases and the three given tasks.

Test case	Track	Topic pred	Change pred
T1	0.749	0.642	0.826
T2	0.727	0.542	0.826
T3	0.798	0.714	0.796
T4	0.738	0.665	0.837
T5	0.800	0.667	0.780
T6	0.752	0.654	0.812
T7	0.666	0.573	0.838
T8	0.755	0.676	0.820
T9	0.728	0.596	0.783
T10	0.744	0.606	0.876
Cross	0.750	0.652	0.818

Table 5.6: Results of our final models evaluated on each of the 10 test cases T1 to T10. The F-scores for each of the tasks are given by the last three columns. The last row contains the cross-validation result which is an approximated overall performance of the models.

Observations can be made that not all test cases are equally difficult, e.g. a high tracking score of 0.800 is achieved for test case 5 while the best score for test case 7 is only 0.666. But for most test cases, the resulting F-scores show similar values. The difficulty of a test case also depends on the task it is trained for. There are test cases that achieve good scores on both the tracking and the next topic prediction tasks, but there are also test cases that perform better on one task than the other. Test case 5 for example has a much higher F-score on the tracking task than test case 4, but their scores on the next topic prediction task are about the same. Although a certain correlation between the scores of the tracking and next topic prediction tasks can be observed, most of them seem to be rather unrelated to the scores for

the topic change prediction task. For example the top 2 scoring test cases 3 and 5 for the tracking task obtain only one of the lower results for the topic change prediction task. Also test case 10, which has rather average scores on the other two tasks, is able to achieve the best F-score for the topic change prediction task. The results for the prediction tasks seem to be influenced by the number of topic changes in the test cases which especially yields for topic change prediction. The cross-validation performance for each of the tasks can be found in the last row of table 5.6. Overall, the best F-score of 0.818 is obtained for the topic change prediction task, which is understandable since it only needs to decide between two classes. The tracking task achieves a higher score of 0.750 compared to 0.652 for the next topic prediction task which demonstrates that making topic predictions is indeed a more challenging task than tracking the current topic.

More detailed results can be found in table 5.8, 5.9 and 5.10, which show the confusion matrix for each of the tasks. For the multi-class tasks tracking and topic prediction, the highest values for each of the topic labels lie on the diagonal of the confusion matrix and most of the other matrix entries are comparably small. There are some topics that often get mixed up for both tasks because of certain similarities in the examples. For instance some of the *Health* examples are classified as *Personal* and vice versa because *Health* is a topic where the speakers also talk a lot about themselves. Hence it is probably very difficult for the models to differentiate between these two in some cases. *ComputerAndInternet* and *Technology* are also topics that share many similarities and thus it is possible that the models are sometimes unable to tell them apart. Other occasional mix-ups mostly also make sense, like it is understandable that an example for *Acquaintance* is classified as *Family* because both are topics about persons.

Figure 5.11 further provides the topic class F-scores and error rates calculated from the confusion matrix for both multi-class tasks. It shows that in most cases, our models are able to determine a majority of the labels for each class correctly. But there are also topics, especially for the topic prediction task, where the error rate is over 50%. All of these topics have only about 50 examples or less which is probably one of the reasons for the bad performance. But there are also topics like *Politics* and *Language* which only have 31 and 41 examples respectively, and achieve top scores among all topic classes. On the other side, topics like *Acquaintance*, *Work* and *Health* which have the most number of examples, are only able to obtain average scores. Hence in addition to the amount of training examples, there is also a difficulty level that influences the performance on the topic classes. While *Politics* and *Language* are special topics where it is easy to find key words, topics like *Acquaintance*, *Work* and *Health* are quite general and cover a lot of subtopics. The lowest scoring classes *ComputerAndInternet*, *Location* and *Personal* are all topics that are either easily mistaken for another or too general for the little amount of training examples.

After further investigation of the test results, we find out that except for semantically similar topic classes, mistakes are also made right before or after a topic change for the tracking task. This occurs because some dialog segments contain a topic change themselves and only get one of the two topic labels assigned. In this case, the model is not always able to choose the right one which is indeed a difficult decision. Although the scores for the next topic prediction task are relatively high, it seems

that the models are still not accurate enough when a topic change occurs. The results for all topic changes in each of the 10 test cases is summarized in table 5.7.

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	$\sum$
Correct topic	9	10	14	7	14	15	12	13	11	7	112
Wrong topic	8	9	12	9	7	10	12	6	7	9	89
No change	10	17	21	11	19	13	15	16	23	14	159
$\sum$	27	36	47	27	40	38	39	35	41	30	360

Table 5.7: Results on topic changes for the next topic prediction task. Correct topic means that the new topic is found correctly whereas wrong topic means that although the change is predicted correctly, but not the new topic. No change means that the model simply returns the topic label from the previous example. The last row contains the overall number of topic changes for each of the test cases.

For example out of the 27 topic changes from test case 1, 9 new topics are predicted correctly, another 8 changes are also predicted correctly but a wrong new topic is chosen and in the last 10 cases, the model just returns the same topic label as for the previous example. So the model is able to make correct topic predictions 33% of the time and overall predict 63% of the changes in test case 1. The comparatively high score for the entire test case is possible because there are not as many topic changes as no changes which are mostly correctly predicted by the model. For the wrongly selected new topics, the model often makes mistakes by choosing a semantically similar topic class to the correct one. There are also many cases where the model returns a new topic label although there is no actual topic change. But a similar behavior can also be observed for the tracking task and it is probably because there are several difficult example segments that the CNNs are not able to find the right labels for. Overall, our models are able to correctly predict 31% of the new topics and 56% of the topic changes from all 10 test cases.

The confusion matrix for the topic change prediction task in table 5.10 further demonstrates the difficulty of the prediction task. Although it is trained to solely predict the topic change, it is only able to return 31% of the changes correctly. It does not achieve better results than models trained on all topic labels and this is probably because the models are trained on many more ”same” examples than ”new” examples. In this case it can even happen that trained models only return ”same” and classify all ”new” examples wrongly. Our models still manage to predict several ”new” examples correctly which is already a pretty good result. But as seen previously as well, making prediction for changes in natural conversations is a much more difficult task compared to tracking. Our results show that although not all new topics and topic changes can be predicted correctly, the models are still able to at least achieve accuracies of over 30%. Furthermore, the models for both prediction tasks are mostly accurate when no topic change takes place.

	Acq	Ani	Car	Clo	Com	Dat	Edu	Fam	Fin	Foo	Hea	Hou	Lan	Loc	Mus	Mov	Per	Pol	Rel	Spo	Tec	Tra	Wor	$\sum$
Acq	**109**	1	0	2	1	0	0	5	3	2	5	2	0	1	3	2	1	0	0	1	0	2	7	147
Ani	4	**26**	0	0	1	0	0	2	0	4	3	3	0	1	0	0	1	0	0	1	0	2	0	48
Car	1	0	**21**	0	0	0	0	1	1	1	1	1	0	0	0	0	1	0	0	1	0	0	1	30
Clo	1	0	0	**38**	0	0	0	1	1	0	0	0	0	1	0	0	1	0	0	1	0	1	0	45
Com	0	1	0	1	**21**	0	1	0	1	0	0	1	0	0	0	1	5	0	0	0	7	1	3	43
Dat	1	0	0	0	0	**24**	2	1	1	0	2	0	0	1	1	1	1	0	0	0	0	1	1	37
Edu	0	0	0	0	0	0	**71**	0	0	0	0	0	3	1	1	1	1	1	0	0	0	1	4	84
Fam	5	1	0	0	0	0	1	**82**	0	0	1	0	1	0	0	0	2	0	0	0	0	0	3	96
Fin	2	0	0	2	0	0	0	2	**66**	1	2	2	0	1	0	0	1	1	0	0	0	0	3	83
Foo	5	1	0	1	0	0	1	1	1	**86**	0	0	0	1	0	1	1	0	0	0	0	0	0	99
Hea	3	3	0	1	0	0	1	5	0	2	**84**	1	0	0	0	3	13	0	0	0	0	0	1	117
Hou	2	1	0	0	0	1	0	1	4	1	0	**49**	1	4	0	0	1	0	0	0	1	1	0	67
Lan	0	0	0	0	0	0	1	1	0	0	0	0	**37**	0	0	0	1	0	0	0	0	0	0	40
Loc	1	1	1	0	1	1	0	1	1	0	0	7	0	**16**	0	1	0	0	0	0	0	4	0	35
Mus	2	0	0	0	0	1	1	0	0	0	0	0	0	0	**53**	1	2	0	0	0	0	0	0	60
Mov	7	1	0	0	0	2	1	0	0	1	3	1	0	0	1	**89**	6	0	0	0	0	0	2	114
Per	2	0	0	0	1	0	4	2	1	0	8	0	0	0	0	1	**49**	0	0	0	1	1	4	74
Pol	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	**22**	0	0	0	0	2	27
Rel	3	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	4	0	**32**	0	0	0	0	44
Spo	1	1	0	1	0	0	3	1	1	0	1	1	0	0	1	1	0	0	1	**29**	0	2	0	44
Tec	1	0	0	0	4	0	2	0	1	0	0	0	0	0	1	1	0	0	0	1	**34**	0	2	47
Tra	0	1	0	0	0	1	0	0	3	1	2	1	0	3	1	0	0	1	0	1	0	**71**	4	90
Wor	1	0	0	0	2	0	4	0	3	3	3	2	0	0	1	1	4	0	0	0	1	3	**101**	129
$\sum$	152	38	22	47	31	31	94	106	89	102	116	72	42	30	63	104	96	25	33	35	44	90	138	1600

Table 5.8: Confusion matrix for the tracking task containing results from all 10 test cases. Each column indicates a predicted topic label and the rows represent actual reference labels. Topic labels are each abbreviated by their first three letters for better view. The sum of each column and row can be found in the last column and row, respectively.

	Acq	Ani	Car	Clo	Com	Dat	Edu	Fam	Fin	Foo	Hea	Hou	Lan	Loc	Mus	Mov	Per	Pol	Rel	Spo	Tec	Tra	Wor	$\sum$
Acq	**91**	0	1	2	0	1	0	8	4	2	5	3	1	2	2	4	6	0	0	1	0	3	9	145
Ani	6	**18**	0	0	2	0	1	3	0	5	3	6	1	2	0	0	0	0	0	1	0	0	0	48
Car	4	0	**16**	1	1	0	1	0	0	0	0	1	0	1	1	0	0	0	0	0	2	1	2	31
Clo	2	0	0	**33**	0	0	2	0	1	2	2	1	0	0	0	0	0	0	0	0	1	1	1	46
Com	0	0	0	0	**17**	0	0	2	0	1	2	2	1	1	1	1	5	0	0	0	6	0	3	42
Dat	2	0	0	1	0	**24**	0	0	0	0	0	0	0	0	1	3	0	0	0	0	1	3	2	37
Edu	0	1	0	1	1	1	**65**	3	1	0	0	0	1	0	2	0	4	0	0	0	0	1	6	87
Fam	8	0	1	0	1	0	1	**71**	1	2	2	2	0	1	0	0	1	0	0	1	0	1	6	99
Fin	2	0	1	0	0	0	0	2	**60**	2	2	5	0	1	1	2	2	2	0	0	0	1	6	89
Foo	1	0	0	1	0	1	2	3	3	**80**	2	2	0	2	0	1	0	0	0	0	1	1	0	100
Hea	7	1	0	3	0	0	2	3	2	4	**81**	2	0	0	0	3	8	0	1	1	0	1	3	122
Hou	1	1	0	0	0	0	1	4	1	3	0	**43**	1	5	0	0	2	1	0	0	1	1	1	66
Lan	0	0	0	0	1	0	1	0	0	1	0	1	**33**	0	0	0	1	0	0	0	0	2	0	40
Loc	1	1	1	0	2	1	1	2	2	0	1	3	1	**12**	0	0	1	0	0	0	1	2	1	33
Mus	3	0	0	0	0	1	1	0	0	1	0	0	0	0	**47**	2	0	0	1	1	1	0	0	58
Mov	8	1	0	0	0	1	0	1	1	1	8	1	0	0	1	**81**	6	1	0	0	1	0	3	115
Per	7	0	0	0	2	0	6	2	0	0	11	0	1	1	0	5	**34**	0	0	0	1	0	3	73
Pol	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**22**	0	0	0	2	1	26
Rel	3	0	0	1	0	0	0	0	1	0	1	2	0	0	1	0	1	1	**31**	0	0	0	0	42
Spo	1	1	2	1	0	0	3	1	2	0	5	1	0	1	1	1	0	0	0	**21**	0	1	1	43
Tec	2	0	1	0	2	1	1	2	2	1	1	1	0	0	1	0	1	0	0	0	**23**	1	5	45
Tra	2	1	0	0	1	2	1	2	2	0	0	2	1	2	0	0	1	1	2	0	0	**67**	3	90
Wor	4	0	0	1	1	1	4	2	6	2	2	1	0	0	0	3	1	0	0	0	2	4	**89**	123
$\sum$	155	25	23	45	31	34	93	111	89	107	128	79	41	31	59	106	75	28	35	26	41	93	145	1600

Table 5.9: Confusion matrix for the next topic prediction task containing results from all 10 test cases. Each column indicates a predicted topic label and the rows represent actual reference labels. Topic labels are each abbreviated by their first three letters for better view. The sum of each column and row can be found in the last column and row, respectively.

	Same	New	$\sum$
Same	**1030**	210	1240
New	247	**113**	360
$\sum$	1277	323	1600

Table 5.10: Confusion matrix for the topic change prediction task containing results from all 10 test cases. Each column indicates a predicted topic label and the rows represent actual reference labels where ”same” means that the topic stays the same and ”new” means that the topic changes to a new one. The sum of each column and row can be found in the last column and row, respectively.

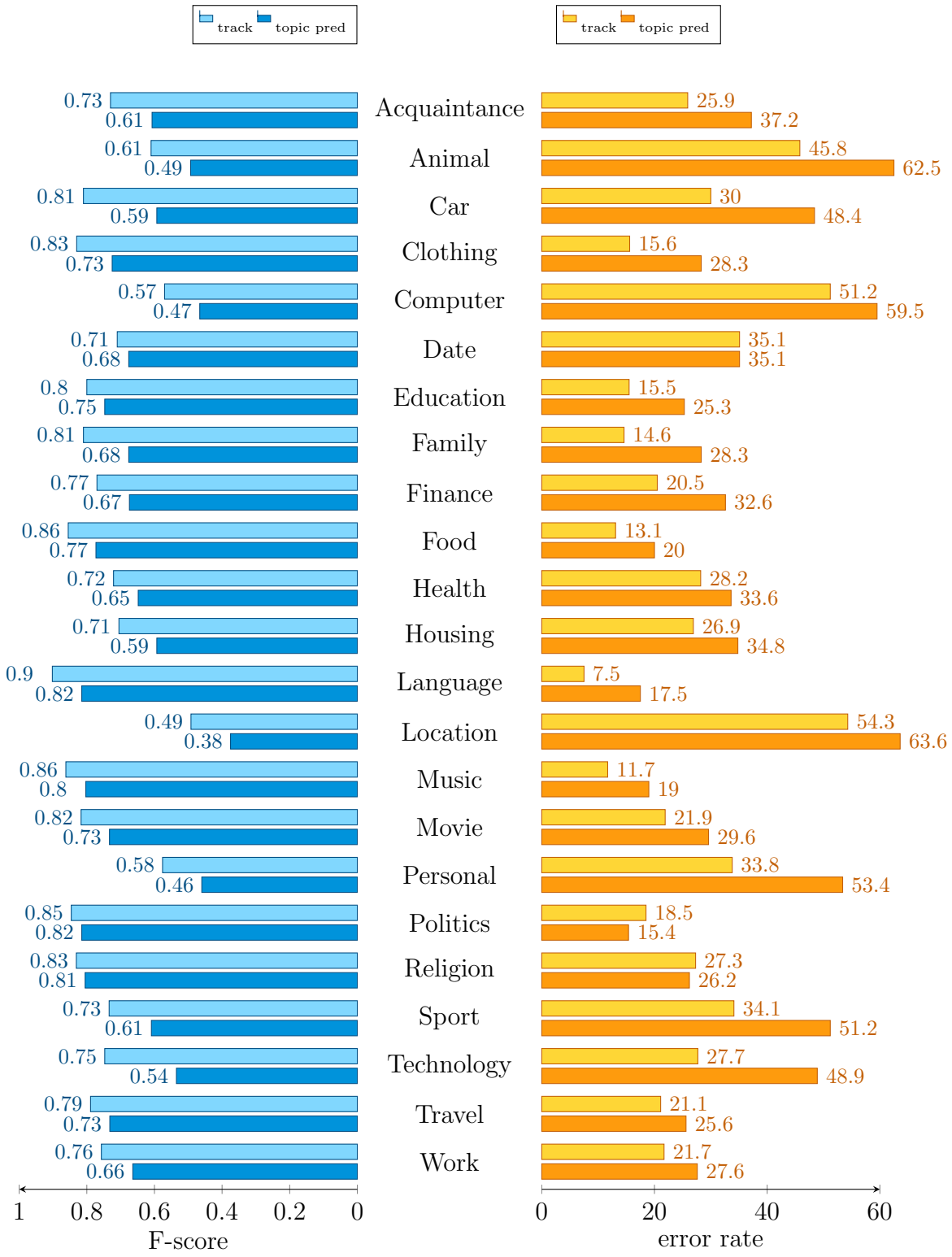

Figure 5.11: Final results for each of the topic labels evaluated on all 10 test cases. Topic classes for both the tracking and the next topic prediction tasks are evaluated with the F-score (left) as well as the error rate (right). Labels consisting of more than one word are abbreviated by their first word.

6. Conclusion

In this work, we have described our approach to solve the tasks of topic tracking, next topic prediction and topic change prediction in non-goal-oriented dialogs. We have provided a formalized definition of dialog state and a detailed formulation of the three given tasks. To solve these tasks, three different CNN architectures are employed which are the basic model, the multilayer network and the word embedding network. The backpropagation algorithm is used to train our models and optimization techniques like dropout, L1 and L2 regularization are further implemented to improve the generalization ability of the models. In addition to CNNs, several word models are trained as well which are used to convert raw words into continuous vector representations. Since there are no labeled dialog corpora available which are suited for our tasks, we have annotated our collected unlabeled data manually with topic labels found during the data creation process.

Our models are evaluated thoroughly on the labeled dialog corpus which is divided into 10 disjoint test cases. At first we discover, that the models can be trained in a short time because a near convergence performance can already be achieved after 10 training epochs. But since continuing training can further obtain significant improvements, it is still beneficial to train these networks until the final stopping criteria are met. We further employ different word models to generate vector representations for our basic model and results show that the overall performance of the CNNs can be significantly increased by using a more accurate word model. After an extensive evaluation of model and training hyperparameters, the optimal values for each of these parameters are found. We also experiment with various settings for the implemented regularization methods, but the performance gains are comparatively small, especially for the L1 and L2 regularizations. For dropout, only larger networks are able to benefit from this scheme, but at the same time, we do not have enough data to train more complex networks and hence have to be satisfied with the small gains. The sparsity of data is also a huge problem for the multilayer and the word embedding models. Both of them are unable to outperform the best basic model, although they should be more powerful due to the extensions in their architectures. We believe that with more training data, the additional parameters of the multilayer model can be trained more effectively. Also for the word embedding architecture, it is highly possible that it can outperform the basic architecture

given sufficient amount of training data. This can be seen through the performance comparison of the word embedding model and the basic model which uses a word model trained on our much smaller dialog corpus.

The final performance achieved by our CNN models is evaluated using the F-score and for all 10 test cases. For the tracking task, our models obtain a cross-validation score of 0.750. A score of 0.652 and 0.818 is achieved for the next topic prediction and the topic change prediction task, respectively. Since there have been no well-known works on topic tracking in non-goal-oriented dialogs before and the data that we use is created by ourselves, there are no existing works to compare our results with. But given the performance of various goal-oriented dialog state trackers, we can say that our approach is able to obtain great results considering the high complexity of the tasks. Especially for the tracking task, our model is fairly accurate, even for some topic classes which are under-represented in the training data. However, the prediction of a topic change seems to be a much more difficult task. Our topic prediction models are only able to correctly predict 31% of the new topics and 56% of the topic changes. Since there are less topic changes than no changes and our models are mostly accurate when the topic does not change, the overall score for the task is comparatively high. Despite the errors, we think that the performance of our networks can still be considered as good. Predicting the topic in a conversation is a task that is difficult even for human and all the more for computers. Given the little amount of training data and the complexity of the tasks, our models are still able to predict several new topics correctly and over half of the topic changes which shows that CNN approaches can be effectively employed for not only tracking but also for making predictions.

As seen in the results, the sparsity of training data is a major issue for training the CNNs. Because of the time limit of this work, it is not possible to create a much larger labeled dialog corpus, but for future works, it would be interesting to evaluate our approach on more labeled conversations. With sufficient amount of training data, it is possible that our multilayer and word embedding architectures can outperform the basic model. Other neural network architectures can be employed for topic prediction as well, for example recurrent networks which have shown great performance on various dialog state tracking tasks. Also hybrid models which combine rule-based and neural network approaches have been successful on this field. By considering the results of such a rule-based model, it is possible that the accuracy of our CNNs can be further improved. Overall, there is still room for improvements on the prediction accuracy for topic changes. This can be potentially achieved by employing other information like using part-of-speech tagging and sentiment analysis in addition to the utterances. Furthermore, since our approach only utilizes raw dialogs which are outputs of the ASR component in a dialog system, it would be interesting to also experiment with outputs of the SLU component.

Appendix

A. BilingBank Files

Corpus	Filename	Speakers (Age)	Description
Miami (eng, spa)	herring01.cha	Lauren (27) Chloe (24)	Two cousins in a restaurant
	herring02.cha	Tomas (19) Miguel (21)	Telephone conversation between two friends
	herring03.cha	Ashley (37) Jack (41)	Conversation in the communal lounge area of an apartment
	herring05.cha	Noah (40) Megan (41)	A couple at their house
	herring06.cha	Jessica (43) Nicholas (-)	A couple at their house
	herring07.cha	Richard (22) Sebastian (-)	Two workmates in a shopping mall café
	herring08.cha	Melanie (39) Robert (42)	A couple in their garden
	herring09.cha	Claire (21) Luke (20)	A couple at Florida International University
	herring10.cha	Paige (33) Sarah (34)	Two co-workers at their workplace
	herring11.cha	Caleb (64) Grace (63)	Conversation between brother and sister
	herring12.cha	Miguel (22) Timothy (20)	Conversation at Miguel's home
	herring13.cha	Leah (-) Vanessa (32)	Conversation between co-workers
	herring14.cha	Gabrielle (23) Connor (20)	Conversation at Florida International University
	herring15.cha	Brandon (-) Evan (21)	Two friends at Florida International University

herring16.cha	Abel (24) Ian (30)	Conversation at Abel's house
herring17.cha	Iris (25) James (-)	Two friends at a café
sastre01.cha	Sofia (44) Kevin (57)	Two neighbours at Kevin's house
sastre02.cha	Ava (78) Luis (55)	Luis and his mother at Luis' place of work
sastre04.cha	Emily (29) Gianna (22)	Two co-workers at their workplace
sastre06.cha	Aaron (43) Alyssa (42)	A couple in their living room
sastre08.cha	Paola (13) Audrey (63)	Paola and her grandmother at a friend's house
sastre09.cha	Kayla (48) Valeria (60)	Two sisters at Kayla's workplace
sastre10.cha	Jocelyn (35) Jennifer (35)	Conversation between two cousins
sastre11.cha	Diego (30) Evelyn (60)	Two friends at the researcher's house
sastre12.cha	Samantha (41) Madeline (48)	Two sisters at Samantha's house
sastre13.cha	Cole (25) Elizabeth (19)	Conversation between friends
zeledon01.cha	Carolina (21) Amelia (26)	Two classmates at Florida International University
zeledon02.cha	Mathew (22) Rebecca (21)	Two classmates at Florida International University
zeledon03.cha	Felipe (11) Elena (19)	Two cousins at Felipe's house
zeledon04.cha	Ethan (40) Henry (-)	Two neighbors at Ethan's house
zeledon05.cha	Maya (37) Isabella (35)	Conversation between two friends
zeledon06.cha	Abigail (21) Ella (19)	Two co-workers at Florida International University
zeledon08.cha	Marcela (45) Flavia (42)	Two neighbors at Marcela's house
zeledon09.cha	Chantal (12) Gillian (9)	Conversation between two cousins

	zeledon11.cha	Sean (25) Antonio (21)	Two friends at Florida International University
	zeledon13.cha	Ariana (18) Avery (19)	Two friends at Florida International University
	zeledon14.cha	Herminia (22) Laurie (19)	Two friends at Florida International University
Pilot (eng, cym)	holly.cha	Alpha (46) Bravo (26)	Conversation at home
Siarad (eng, cym)	davies2.cha	Greta (23) Gwylan (23)	Two close friends in a University meeting room
	davies3.cha	Tostig (15) Harold (13)	Conversation recorded at the University
	davies17.cha	Glain (35) Robin (31)	Conversation between friends

Table A.1: All files from the BilingBank corpora that are labeled and used as the training dataset. These corpora are collected and transcribed at the Bangor University in Wales. The Miami corpus contains informal conversations between English-Spanish bilinguals and the Pilot and the Siarad corpus are both recorded from English-Welsh speakers.

Test case	Files
Test case 1	herring01.cha, herring16.cha, sastre11.cha(1), zeledon11.cha, davies17.cha
Test case 2	herring15.cha, sastre02.cha, sastre12.cha, zeledon01.cha, zeledon06.cha
Test case 3	herring06.cha, sastre08.cha, sastre13.cha, davies2.cha
Test case 4	herring11.cha(1), sastre01.cha, zeledon02.cha, zeledon09.cha
Test case 5	herring02.cha, herring05.cha, herring07.cha(2), sastre04.cha, zeledon04.cha
Test case 6	herring03.cha, herring12.cha, herring17.cha(1), sastre06.cha, holly.cha
Test case 7	herring08.cha, herring09.cha, herring11.cha(2), herring17.cha(2), sastre09.cha(1)
Test case 8	herring13.cha, sastre09.cha(2), zeledon05.cha, zeledon08.cha
Test case 9	herring14.cha, sastre10.cha(1), sastre11.cha(2), zeledon03.cha, zeledon14.cha
Test case 10	herring07.cha(1), herring10.cha, sastre10.cha(2), zeledon13.cha, davies3.cha

Table A.2: The labeled dataset is partitioned into 10 test cases with 4 or 5 dialog sessions each. There are a few very long conversations which are divided in half for a better distribution of topics. This is indicated by the numbers in brackets where (1) is the first half and (2) the second half of the dialog file.

B. Topic Labels

Topic label	Subtopics
Acquaintance	Friend, person that the speaker knows well
AnimalAndPlant	Pet, garden plant
Car	Passenger car, truck
Clothing	Clothes, accessory
ComputerAndInternet	Computer equipment, website
DateAndTime	Schedule, appointment
Education	School, college, homework
Family	Member of the family
Finance	Budget, cost
FoodAndDrink	Food, drink, restaurant
Health	Health issue, medical treatment
Housing	House, room, furniture
Language	Characteristic of languages and words
Location	Place, directions
MovieAndTV	Movie, television program
Music	Classic and modern music
Personal	Speaker's characteristic, mood and feeling
Politics	Election, party, politician
Religion	Religious belief and activity
Sport	Sports, training, fitness
Technology	Modern technology, electronic device
Travel	Transportation, vacation spot
Work	Work-related task, workplace, colleague

Table B.1: Overview of the 23 topic labels found during the labeling process. The right column contains examples of associated subtopics which appear in the training dataset.

C. Word2vec Results

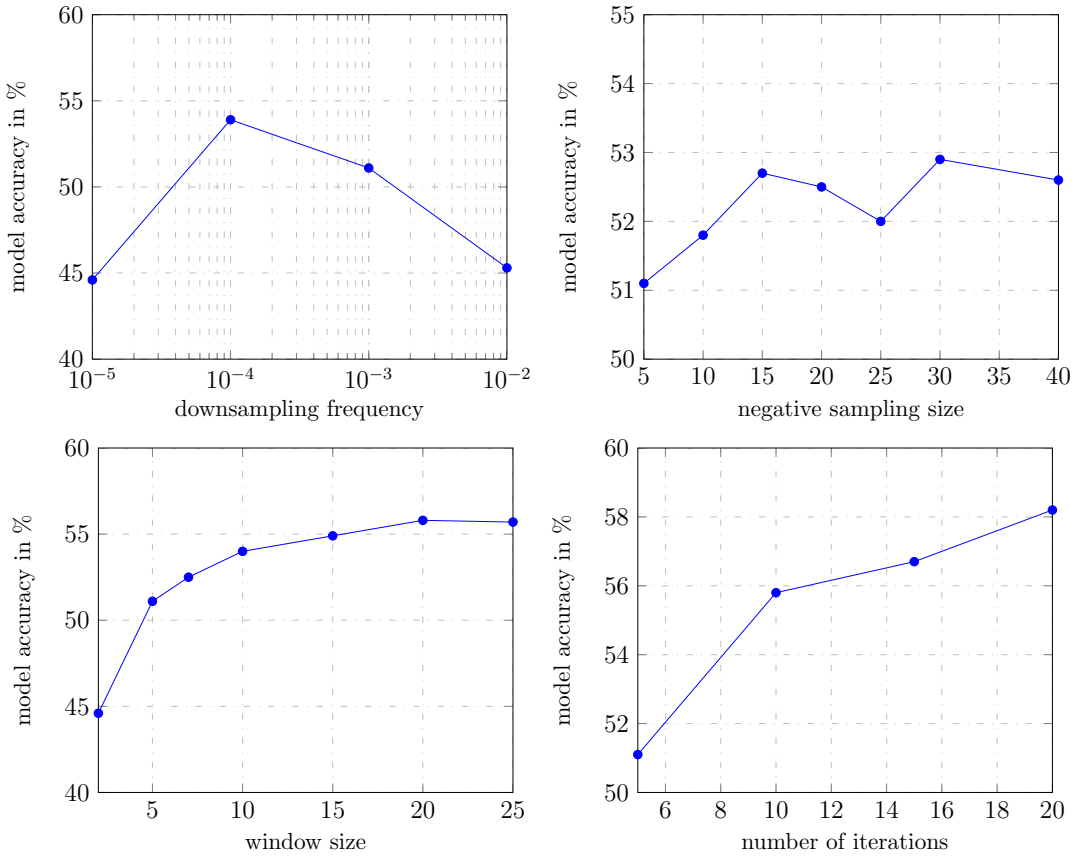

Figure C.1: Overall accuracy of word models trained with one varying parameter while leaving the remaining ones at their default values. Top left shows models which were trained with different downsampling frequency thresholds, so that words with higher frequency in the training data are randomly downsampled. Top right shows models trained with different number of noise words that should be drawn for negative sampling. The models in bottom left were trained on various window sizes which is the maximum distance between the current and the predicted word within a sentence. Bottom right displays models trained with increasing number of iterations over the corpus.

Category	Example	Accuracy
capital-common-countries	Athens Greece Berlin Germany	93.3%
capital-world	Abuja Nigeria Accra Ghana	83.0%
currency	Argentina peso USA dollar	7.8%
city-in-state	Chicago Illinois Dallas Texas	60.9%
family	brother sister son daughter	91.2%
gram1-adjective-adverb	calm calmly slow slowly	29.5%
gram2-opposite	aware unaware clear unclear	46.0%
gram3-comparative	bad worse big bigger	79.3%
gram4-superlative	bad worst big biggest	77.6%
gram5-present-participle	code coding debug debugging	67.2%
gram6-nation-adjective	Brazil Brazilian Germany German	97.6%
gram7-past-tense	falling fell going went	63.4%
gram8-plural	banana bananas bird birds	84.3%
gram9-plural-verbs	eat eats speak speaks	54.9%
newspapers	Denver Denver_Post Boston Boston_Globe	100%
ice_hockey	Dallas Dallas_Stars Boston Boston_Bruins	100%
airlines	Germany Lufthansa Canada Air_Canada	100%
people-companies	Tim_Cook Apple Bill_Gates Microsoft	50.0%

Table C.1: Overview of all categories from Google's test data evaluated on the word2vec model. The accuracy column contains results achieved by the overall best word model trained with size = 200, alpha = 0.05, window = 8, sample = 0.0001, negative = 25 and iter = 15.

D. Parameter Performances

Batch size	Track	Topic pred	Change pred
10	0.62	0.54	0.77
25	0.63	0.53	0.79
50	**0.68**	**0.57**	**0.84**

History length	Track	Topic pred	Change pred
0	0.70	**0.61**	**0.82**
1	**0.72**	0.56	**0.82**
2	0.65	0.46	0.70

Activation function	Track	Topic pred	Change pred
tanh	**0.70**	**0.61**	0.82
ReLu	0.66	**0.61**	**0.83**

Table D.1: Performance comparison of different values for the parameters batch size, history length and activation function. Models for the batch size tests are trained with size 1, 2, 3 and 4 filters, and 200 filters each. For the other two parameters, the models consist of 300 filters with size 1, 2 and 3. The last three columns contain the F-scores for the tracking, next topic prediction and topic change prediction tasks, respectively.

Bibliography

[BeDFF10] A. C. Berg, J. Deng and L. Fei-Fei. Large Scale Visual Recognition Challenge 2010, 2010.

[Bish06] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer. 2006.

[Blac05] G. W. Blackwood. Neural Network-based Language Model for Conversational Telephone Speech Recognition. Master's Thesis, St. Catherine's College, Oxford, England, July 2005.

[BoRu06] D. Bohus and A. Rudnicky. A 'K Hypotheses + Other' Belief Updating Model. In *Proceedings AAAI Workshop on Statistical and Empirical Approaches for Spoken Dialogue Systems*, Boston, USA, 2006.

[CaVGB06] P. L. Callet, C. Viard-Gaudin and D. Barba. A Convolutional Neural Network Approach for Objective Video Quality Assessment. *IEEE Transactions on Neural Networks*, 17(5), September 2006, pp. 1316–1327.

[CiMS12] D. Cireşan, U. Meier and J. Schmidhuber. Multi-column Deep Neural Networks for Image Classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, June 2012, pp. 3642–3649.

[CMML+11] D. Cireşan, U. Meier, J. Masci, L.Gambardella and J. Schmidhuber. Flexible, High Performance Convolutional Neural Networks for Image Classification. In *International Joint Conference on Artificial Intelligence*, July 2011, pp. 1237–1242.

[CMMS12] D. Cireşan, U. Meier, J. Masci and J. Schmidhuber. Multi-column Deep Neural Network for Traffic Sign Classification. *Neural Networks*, 32, 2012, pp. 333–338.

[CoWe08] R. Collobert and J. Weston. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proceedings of the 25th International Conference on Machine Learning*, New York, NY, USA, 2008, pp. 160–167.

[CWBK+11] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu and P. P. Kuksa. Natural Language Processing (almost) from Scratch. *Computing Research Repository*, abs/1103.0398, 2011.

[DeSt07] D. DeVault and M. Stone. Managing Ambiguities Across Utterances in Dialogue. In *Proceedings Workshop on the Semantics and Pragmatics of Dialogue*, Trento, Italy, 2007.

[DLBB16] F. Dernoncourt, J. Y. Lee, T. H. Bui and H. H. Bui. Robust Dialog State Tracking for Large Ontologies. *Computing Research Repository*, abs/1605.02130, 2016.

[DNMLe11] C. Danescu-Niculescu-Mizil and L. Lee. Chameleons in Imagined Conversations: A New Approach to Understanding Coordination of Linguistic Style in Dialogs. In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics (ACL 2011)*, 2011.

[FaCl54] B. Farley and W. Clark. Simulation of Self-organizing Systems by Digital Computer. *Transactions of the IRE Professional Group on Information Theory*, 4(4), September 1954, pp. 76–84.

[Fell05] C. Fellbaum. Wordnet and Wordnets. In *Encyclopedia of Language and Linguistics*, Oxford, 2005, pp. 665–670.

[GBFH14] E. Grefenstette, P. Blunsom, N. de Freitas and K. M. Hermann. A Deep Architecture for Semantic Parsing. *Computing Research Repository*, abs/1404.7296, 2014.

[GlBB11] X. Glorot, A. Bordes and Y. Bengio. Deep Sparse Rectifier Neural Networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, 15. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011, pp. 315–323.

[GlBe10] X. Glorot and Y. Bengio. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *Aistats*, 9, May 2010, pp. 249–256.

[GoBC16] I. Goodfellow, Y. Bengio and A. Courville. *Deep Learning*. MIT Press. http://www.deeplearningbook.org, 2016.

[GrSc09] A. Graves and J. Schmidhuber. Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks. In *Advances in Neural Information Processing Systems 22*, Vancouver, BC, December 2009, pp. 545–552.

[Hend15] M. Henderson. Machine Learning for Dialog State Tracking: A Review. In *Proceedings of The First International Workshop on Machine Learning in Spoken Language Processing*, 2015.

[HeTW14a] M. Henderson, B. Thomson and J. Williams. The Second Dialog State Tracking Challenge. In *Proceedings of the SIGDIAL 2014 Conference*, 2014.

[HeTW14b] M. Henderson, B. Thomson and J. Williams. The Third Dialog State Tracking Challenge. In *Proceedings IEEE Spoken Language Technology Workshop*, December 2014.

[HeTY13] M. Henderson, B. Thomson and S. Young. Deep Neural Network Approach for the Dialog State Tracking Challenge. In *Proceedings of the SIGDIAL 2013 Conference*, Metz, France, August 2013, pp. 467–471.

[HeTY14a] M. Henderson, B. Thomson and S. Young. Robust Dialog State Tracking Using Delexicalised Recurrent Neural Networks and Unsupervised Adaptation. *Proceedings IEEE Workshop on Spoken Language Technology*, 2014.

[HeTY14b] M. Henderson, B. Thomson and S. Young. Word-Based Dialog State Tracking with Recurrent Neural Networks. In *Proceedings SIGDIAL Conference on Discourse and Dialogue*, Philadelphia, USA, January 2014, pp. 292–299.

[HiNA03] R. Higashinaka, M. Nakano and K. Aikawa. Corpus-based Discourse Understanding in Spoken Dialogue Systems. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, Stroudsburg, PA, USA, 2003, pp. 240–247.

[HSKS+12] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. Salakhutdinov. Improving Neural Networks by Preventing Co-adaptation of Feature Detectors. *Computing Research Repository*, abs/1207.0580, 2012.

[HSMD+00] R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas and H. S. Seung. Digital Selection and Analogue Amplification Coexist in a Cortex-inspired Silicon Circuit. In *Nature*, 405, June 2000, pp. 947–951.

[IvLa67] A. G. Ivakhnenko and V. G. Lapa. *Cybernetics and Forecasting Techniques.* American Elsevier Pub. Co. 1967.

[JHLC+16] Y. Jang, J. Ham, B.-J. Lee, Y. Chang and K.-E. Kim. Neural Dialog State Tracker for Large Ontologies by Attention Mechanism. *Proceedings IEEE Workshop on Spoken Language Technology*, 2016, pp. 531–537.

[JoZh14] R. Johnson and T. Zhang. Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. *Computing Research Repository*, abs/1412.1058, 2014.

[JoZh15] R. Johnson and T. Zhang. Semi-supervised Convolutional Neural Networks for Text Categorization via Region Embedding. In *Advances in Neural Information Processing Systems 28*, pp. 919–927. Curran Associates, Inc., 2015.

[JXYY13] S. Ji, W. Xu, M. Yang and K. Yu. 3D Convolutional Neural Networks for Human Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1), January 2013, pp. 221–231.

[KaGB14] N. Kalchbrenner, E. Grefenstette and P. Blunsom. A Convolutional Neural Network for Modelling Sentences. *Computing Research Repository*, abs/1404.2188, 2014.

[KDBW+16a] S. Kim, L. F. D'Haro, R. E. Banchs, J. Williams and M. Henderson. The Fourth Dialog State Tracking Challenge. In *Proceedings International Workshop on Spoken Dialog Systems*, January 2016.

[KDBW+16b] S. Kim, L. F. D'Haro, R. E. Banchs, J. Williams, M. Henderson and K. Yoshino. The Fifth Dialog State Tracking Challenge. In *Proceedings IEEE Spoken Language Technologies Workshop*, December 2016.

[KiBa14] S. Kim and R. E. Banchs. Sequential Labeling for Tracking Dynamic Dialog States. In *Proceedings SIGDIAL Conference on Discourse and Dialogue*, Philadelphia, PA, U.S.A., June 2014, pp. 332–336.

[Kim14] Y. Kim. Convolutional Neural Networks for Sentence Classification. *Computing Research Repository*, arXiv:1408.5882, 2014.

[KrSH12] A. Krizhevsky, I. Sutskever and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, 2012, pp. 1097–1105.

[KTSL+14] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar and L. Fei-Fei. Large-Scale Video Classification with Convolutional Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 2014, pp. 1725–1732.

[KVLM+15] R. Kadlec, M. Vodolan, J. Libovicky, J. Macek and J. Kleindienst. Knowledge-based Dialog State Tracking. *Proceedings IEEE Workshop on Spoken Language Technology*, April 2015, pp. 348–353.

[LaTr00] S. Larsson and D. R. Traum. Information State and Dialogue Management in the TRINDI Dialogue Move Engine Toolkit. *Natural Language Engineering*, 6(3-4), September 2000, pp. 323–340.

[LBOM12] Y. LeCun, L. Bottou, G. B. Orr and K. Müller. Efficient Backprop. In *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 9–48.

[LeBH15] Y. LeCun, Y. Bengio and G. Hinton. Deep Learning. In *Nature*, 521, May 2015, pp. 436–444.

[LeEs13] S. Lee and M. Eskenazi. Recipe For Building Robust Spoken Dialog State Trackers: Dialog State Tracking Challenge System Description. In *Proceedings of the SIGDIAL 2013 Conference*, Metz, France, August 2013, pp. 414–422.

[LGTB97] S. Lawrence, C. L. Giles, A. C. Tsoi and A. D. Back. Face Recognition: A Convolutional Neural Network Approach. *IEEE Transactions on Neural Networks*, 8(1), 1997, pp. 98–113.

[LiCY14] M. Lin, Q. Chen and S. Yan. Network in Network. In *International Conference on Learning Representations*, 2014.

[LiWu17] M. Li and J. Wu. *The MSIIP System for Dialog State Tracking Challenge 4*, pp. 465–474. 2017.

[LPSP15] R. Lowe, N. Pow, I. Serban and J. Pineau. The Ubuntu Dialogue Corpus: A Large Dataset for Research in Unstructured Multi-Turn Dialogue Systems. *Computing Research Repository*, arXiv:1506.08909, July 2015.

[MacW07] B. MacWhinney. The TalkBank Project. In *Creating and Digitizing Language Corpora: Synchronic Databases*, 1, 2007, pp. 163–180.

[MCCD13] T. Mikolov, K. Chen, G. Corrado and J. Dean. Efficient Estimation of Word Representations in Vector Space. *Computing Research Repository*, arXiv:1301.3781, 2013.

[McPi43] W. S. McCulloch and W. Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics*, 5, December 1943, pp. 115–133.

[MeBW13] A. Metallinou, D. Bohus and J. Williams. Discriminative State Tracking for Spoken Dialog Systems. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, 2013, pp. 466–475.

[MiPa69] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press. 1969.

[MMMK03] M. Matsugu, K. Mori, Y. Mitari and Y. Kaneda. Subject Independent Facial Expression Recognition with Robust Face Detection Using a Convolutional Neural Network. *Neural Networks*, 16(5), 2003, pp. 555–559.

[MSCC+13] T. Mikolov, I. Sutskever, K. Chen, G. Corrado and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. *Advances in Neural Information Processing Systems*, arXiv:1310.4546, 2013.

[RDSK+14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge, 2014.

[ŘeSo10] R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta, May 2010, ELRA, pp. 45–50.

[ReXY14] H. Ren, W. Xu and Y. Yan. Markovian Discriminative Modeling for Cross-domain Dialog State Tracking. *Proceedings IEEE Workshop on Spoken Language Technology*, 2014.

[RHHD56] N. Rochester, J. Holland, L. Haibt and W. Duda. Tests on a Cell Assembly Theory of the Action of the Brain Using a Large Digital Computer. *IRE Transactions on Information Theory*, 2(3), September 1956, pp. 80–93.

[Rose58] F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review*, 1958, pp. 65–386.

[RuHW86] D. E. Rumelhart, G. E. Hinton and R. J. Williams. Learning Representations by Back-propagating Errors. In *Nature*, 323, 1986, pp. 533–536.

[SaGa14] C. D. Santos and M. Gatti. Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts. In *25th International Conference on Computational Linguistics*, Dublin, Ireland, August 2014, pp. 69–78.

[SaZa14] C. D. Santos and B. Zadrozny. Learning Character-level Representations for Part-of-Speech Tagging. In *Proceedings of the 31st International Conference on Machine Learning*, 32, Bejing, China, June 2014, pp. 1818–1826.

[Schm93] J. Schmidhuber. System Modeling and Optimizationn. Habilitation Thesis, Technical University of Munich, Munich, Germany, April 1993.

[SCZY14a] K. Sun, L. Chen, S. Zhu and K. Yu. A Generalized Rule-based Tracker for Dialogue State Tracking. *Proceedings IEEE Workshop on Spoken Language Technology*, 2014.

[SCZY14b] K. Sun, L. Chen, S. Zhu and K. Yu. The SJTU System for Dialog State Tracking Challenge 2. In *Proceedings SIGDIAL Conference on Discourse and Dialogue*, Philadelphia, PA, U.S.A., June 2014, pp. 318–326.

[SEZM+13] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus and Y. LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. In *Computer Vision and Pattern Recognition*, 2013.

[SHGD+14] Y. Shen, X. He, J. Gao, L. Deng and G. Mesnil. Learning Semantic Representations Using Convolutional Neural Networks for Web Search. April 2014, pp. 1725–1732.

[SHKS+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15, 2014, pp. 1929–1958.

[SiZi14a] K. Simonyan and A. Zisserman. Two-Stream Convolutional Networks for Action Recognition in Videos. *Computing Research Repository*, abs/1406.2199, 2014.

[SiZi14b] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-scale Image Recognition. In *Computer Vision and Pattern Recognition*, 2014.

[SLJS+14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich. Going Deeper with Convolutions. In *Computer Vision and Pattern Recognition*, 2014.

[Smol86] P. Smolensky. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. pp. 194–281. MIT Press, Cambridge, MA, USA, 1986.

[SUEY+17a] H. Shi, T. Ushio, M. Endo, K. Yamagami and N. Hori. *Convolutional Neural Networks for Multi-topic Dialog State Tracking*, pp. 451–463. 2017.

[SUEY+17b] H. Shi, T. Ushio, M. Endo, K. Yamagami and N. Horii. Convolutional Neural Networks for Multi-topic Dialog State Tracking. In *Lecture Notes in Electrical Engineering*, 427. Springer Singapore, 2017, pp. 451–463.

[SUEY+17c] H. Shi, T. Ushio, M. Endo, K. Yamagami and N. Horii. A Multichannel Convolutional Neural Network For Cross-language Dialog State Tracking. *Computing Research Repository*, abs/1701.06247, 2017.

[SuLW16] Y. Su, M. Li and J. Wu. The MSIIP System for Dialog State Tracking Challenge 5. *Proceedings IEEE Workshop on Spoken Language Technology*, 2016.

[Tech15] Technology Review. The Face Detection Algorithm Set To Revolutionize Image Search, February 2015.

[Thea16] Theano Development Team. Theano: A Python Framework for Fast Computation of Mathematical Expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[ToPo89] D. S. Touretzky and D. A. Pomerleau. What's Hidden in the Hidden Units? In *BYTE*, 14:8, August 1989, pp. 227–233.

[WaLe13] Z. Wang and O. Lemon. A Simple and Generic Belief Tracking Mechanism for the Dialog State Tracking Challenge: On the Believability of Observed Information. In *Proceedings of the SIGDIAL 2013 Conference*, Metz, France, August 2013, pp. 423–432.

[Werb75] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University. 1975.

[WHHS+89] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano and K. J. Lang. Phoneme Recognition Using Time-delay Neural Networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(3), March 1989, pp. 328–339.

[Will13] J. Williams. Multi-Domain Learning and Generalization In Dialog State Tracking. In *Proceedings of the SIGDIAL 2013 Conference*, Metz, France, August 2013.

[Will14] J. Williams. Web-Style Ranking and SLU Combination For Dialog State Tracking. In *Proceedings of SIGDIAL*, June 2014.

[WiRH16] J. Williams, A. Raux and M. Henderson. The Dialog State Tracking Challenge Series: A Review. *Dialogue and Discourse*, April 2016.

[WiYo07] J. D. Williams and S. Young. Partially Observable Markov Decision Processes for Spoken Dialog Systems. *Computer Speech and Language*, 21(2), April 2007, pp. 393–422.

[WRRB13] J. D. Williams, A. Raux, D. Ramachadran and A. Black. The Dialog State Tracking Challenge. In *Proceedings of the SIGDIAL 2013 Conference*, Metz, France, August 2013.

[WuGu15] H. Wu and X. Gu. Towards Dropout Training for Convolutional Neural Networks. *Neural Networks : The Official Journal of the International Neural Network Society*, 71, July 2015, pp. 1–10.

[WXXL+15] P. Wang, J. Xu, B. Xu, C. Liu, H. Zhang, F. Wang and H. Hao. Semantic Clustering and Convolutional Neural Network for Short Text Categorization. In *Association for Computational Linguistics*, 2015.

[YHNN17] K. Yoshino, T. Hiraoka, G. Neubig and S. Nakamura. Dialogue State Tracking using Long Short Term Memory Neural Networks, 2017.

[ZeFe14] M. D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. In *European Conference on Computer Vision*, 2014, pp. 818–833.

[ZhLe15] X. Zhang and Y. LeCun. Text Understanding from Scratch. *Computing Research Repository*, abs/1502.01710, 2015.

[ZhWa15] Y. Zhang and B. C. Wallace. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. *Computing Research Repository*, abs/1510.03820, 2015.

[ZhZL15] X. Zhang, J. J. Zhao and Y. LeCun. Character-level Convolutional Networks for Text Classification. *Computing Research Repository*, abs/1509.01626, 2015.