Master Thesis

LANGUAGE MODELING WITH RECURRENT NEURAL NETWORKS

Using Transfer Learning to Perform Radiological Sentence Completion

anna-lena popkes

First Supervisor:
Dr. Asja Fischer

Second Supervisor:
Prof. Dr. Christian Bauckhage

Master of Science
Institute of Computer Science
Rheinische Friedrich-Wilhelms-Universität Bonn

March 2018

ABSTRACT

Motivated by the potential benefits of a system that accelerates the process of writing radiological reports, we present a Recurrent Neural Network Language Model for modeling radiological language. We show that Recurrent Neural Network Language Models can be used to produce convincing radiological reports and investigate how their performance can be improved by using advanced regularization techniques like embedding dropout or weight tying, and advanced initialization techniques like pre-trained word embeddings. Furthermore, we study the use of transfer learning to create topic-specific language models. To test the applicability of our techniques to other domains we perform experiments on a second dataset, consisting of forum posts on motorized vehicles. In addition to our experiments on Recurrent Neural Network Language Models, we train a Continuous Bag-of-Words model on the radiological dataset and analyze the resulting medical word embeddings. We show that the embeddings encode medical relationships, semantic similarities and that certain medical relationships can be represented as linear translations.

We can only see a short distance ahead,
but we can see plenty there that needs to be done.

— Alan Turin, 1950

ACKNOWLEDGMENTS

There are several people I want to thank for supporting me in the compilation of this thesis.

First and foremost, my parents who have always supported me in every aspect of my life. I am enormously grateful for all your love, unconditional support and inspiration. Words can not explain how proud I am for having parents like you.

Secondly, I have to thank my supervisors Dr. Asja Fischer and Prof. Dr. Christian Bauckhage. During the last months, Asja was always available for advice and discussions whenever I needed them. Her expertise in Deep Learning helped me enormously and I want to thank her very much for all the time she created. Clearly, this thesis would be a different one without her.

Prof. Bauckhage is one of the best professors I have met during my time at university and I have learned a lot in his lectures. He was also the one who encouraged me to apply for a position at the Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS) where I have been welcomed warmly into a wonderful team of experienced data scientists.

I also want to thank Sven Giesselbach, my friend and colleague at the IAIS. Like Asja, Sven was always available for discussions, and his expertise in Natural Language Processing complemented Asja's expertise in Deep Learning perfectly.

Furthermore, I want to thank Julia Eylers, my brilliant English teacher to-be, who assisted with all incoming questions regarding word order, punctuation and tense.

Last but not least, I thank Pascal Wenker for open ears, valuable advice and never-ending emotional support.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

BPTT	Backpropagation Through Time
CBOW	Continuous Bag-of-Words
CPU	Central Processing Unit
ED	Embedding Dropout
GPU	Graphical Processing Unit
LDA	Latent Dirichlet Allocation
LSA	Latent Semantic Analysis
LSTM	Long Short-Term Memory
MLE	Maximum Likelihood Estimation
NCE	Noise Contrastive Estimation
NEG	Negative sampling
NLP	Natural Language Processing
PCA	Principal Component Analysis
PP	Perplexity
PTB	Penn Treebank
RNNLM	Recurrent Neural Network Language Model
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
VD	Variational Dropout

1 INTRODUCTION

Machine learning, a subdiscipline of artificial intelligence, studies the construction and application of algorithms that are able to learn from data. We can distinguish between three different learning paradigms. In *supervised learning* the training data encountered by an algorithm is *labeled*. This means that for each input value, the machine learning system is also provided with the desired target value. The goal of a supervised system is to learn a mapping from inputs to outputs given a labeled dataset. After learning, the estimated mapping can be used to make predictions for previously unobserved inputs. The term *supervised* originates from the idea that the target value for each input example is provided by a *teacher* that tells the system what to do [18]. In *unsupervised learning* the training examples are *unlabeled*. Consequently, the algorithm has to learn useful properties of the structure of the dataset *without* the help of a teacher. The objective of an unsupervised system is to construct a model or representation of the dataset. The learned model can, for example, be used to make predictions about future inputs. The third form of learning is *reinforcement learning*. Different to supervised and unsupervised learning, reinforcement learning algorithms do not just experience a fixed dataset. Instead, they interact with an *environment* in order to learn to perform some task. A more detailed introduction to supervised, unsupervised and reinforcement learning can be found in Goodfellow et al. [18] and Sutton and Barto [65].

Since their introduction, machine learning algorithms have successfully been applied to a wide variety of fields and tasks, including speech recognition [22], object detection [66] and robotics [33]. In the last years, and with the increasing number of electronic medical records, their application to medical patient data has raised a lot of attention. For instance, machine learning methods were reported to perform cancer detection [37], improve patient care [17, 62] and make medical diagnoses [34].

A large part of all electronic medical records are radiological reports. For example, in England alone 1.76 million plain radiographs were reported for August 2017. Radiology is a branch of medicine that uses medical imaging techniques for the diagnoses and treatment of diseases [52]. The resulting medical images are analyzed by radiologists and their observations and findings are recorded in a written report. These radiological reports typically contain information about the presence or absence of radiological abnormalities, relevant technical factors and references to previous medical examinations [10]. Because the analysis of medical images requires expertise and experience, all reports are written by radiologists. This procedure is time intensive because each report first needs to be written by a radiologist and afterwards proof-read and accepted by another radiologist.

In machine learning, the task of language modeling is central to both Natural Language Processing (NLP) and Language Understanding [55]. A language model assigns probabilities to sequences of words. As pointed out by Jozefowicz et al. [29]: "Models which can accurately place distributions over sentences not only encode complexities of language such as grammatical structure, but also distill a fair amount of information about the knowledge that a corpora may contain". In the last years, especially language models based on Recurrent Neural Networks (RNNs) were found to be effective. After a Recurrent Neural Network Language Model (RNNLM) has been trained on a corpus of text, it can be used to predict the next most likely words in a sequence and thereby generate entire paragraphs of text. Further, RNNLMs have been proven to be able to capture long-term dependencies in data. This is especially important for NLP because sentences may be very long and dependencies may span across multiple sentences.

The work presented in this thesis is motivated by the potential benefits of an RNNLM that supports radiologists in writing radiological reports. A system that dynamically predicts the next word (or words) while writing would not only save huge amounts of time and effort, but could be extended to a wide variety of medical domains. In non-medical domains such systems have already been applied successfully, for instance, to perform autocompletion on smartphones. The time saved by employing an RNNLM in the clinical context could be spend in more meaningful ways. For example, more time for face-to-face interaction with patients would remain. A further advantage of an autocompletion system concerns the language of radiological reports. Although the reports should be written in a comprehensive and systematic fashion, the reporting style can vary a lot between different radiologists [54]. Also, as already pointed out by Cornegruta et al. [10]: "The same findings can often be described in a multitude of different ways". In addition, the reports contain grammar and spelling mistakes. These problems would, to some extend, be solved by the employment of an autocompletion tool. First, the suggested words could be restricted such that spelling mistakes were not allowed or corrected automatically. Second, synonyms and abbreviations for diseases, body parts, medical procedures, etc., could be mapped onto a single term.[1] This would help to standardize reports and make it easier for non-physicians to understand them.

In this thesis we present a word-based RNNLM based on Long Short-Term Memory (LSTM) units for modeling radiological language and investigate its ability to produce sound and complete radiological reports. In our experiments, we study the effects of different regularization and initialization techniques, including the use of pre-trained word embeddings, reusing the input word embedding matrix as the output projection matrix and employing embedding dropout. Most of these recently introduced techniques were reported to improve the performance of RNNLMs significantly [28, 39, 40, 51]. Furthermore, we investigate the use of transfer learning to create topic-specific language models.

To test the applicability of our model to other domains, we perform experiments not only with a large corpus of radiological reports, but also with a corpus from a completely different domain, namely forum posts from the biggest German-language forum on motorized vehicles. In addition to our experiments on RNNLMs, we train a Continuous Bag-of-Words (CBOW) model on the radiological dataset and analyze the resulting medical word embeddings.

This thesis is organized as follows. In Section 2 we introduce the foundations our work builds on. Starting with a general introduction to the topic of language modeling and word embeddings, we further introduce important concepts related to neural language models, and novel regularization techniques that have been proposed in recent years. We end with a discussion about the different ways in which language models can be evaluated. Section 3 introduces relevant related work, both for the task of Recurrent Neural Language Modeling and the use of transfer learning in RNNLMs. Section 4 gives an in-depth overview of the experiments we performed, including details about our model architecture, training procedure and a thorough discussion of the results. In the end, we give a conclusion and outlook in Section 5.

1 This could be done, for example, by including only one term in the vocabulary but not its synonyms.

2

FOUNDATIONS

2.1 STATISTICAL LANGUAGE MODELING

Statistical language modeling is one of the central tasks in NLP and of high importance to several language technology applications, like speech recognition or machine translation [55].
The goal of a statistical language model is to learn the probability $P(w_1, \ldots, w_m)$ of a sequence of words $w_1, \ldots, w_m$ [19, 29, 44]. This probability can be computed using the chain rule of probability:

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \tag{2.1}$$

Because the number of words that precede a word varies, and because it is difficult to compute $P(w_i \mid w_1, \ldots, w_{i-1})$ for a large number of words i, we typically condition the probability of a word on a window of n previous words[1]:

$$P(w_1, \ldots, w_m) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-n}, \ldots, w_{i-1}) \tag{2.2}$$

A language model has several applications. For instance, it can be used to predict upcoming words or to assign probabilities to sentences. This is illustrated by the following example [30]: a language model could predict that the sentence

all of a sudden I noticed three guys standing on the sidewalk

has a higher probability to appear in a text than the same sentence with a different word order:

on guys all I of notice sidewalk three a sudden standing the

This is, inter alia, useful for tasks that require recognizing words in ambiguous and/or noisy input, such as speech recognition.

1 This formula holds for $i > n$. If $i \leqslant n$ we have $P(w_i|w_1, ..., w_{i-1})$.

2.2 N-GRAM MODELS

One of the simplest language models is the N-gram model. An N-gram is a sequence of N words. For example, a bigram (or 2-gram) consists of two words, such as "three guys" or "on the", while a trigram (or 3-gram) consists of three words, like "three guys standing". In case of a trigram model, the probability of a sequence of words $w_1, \dots, w_m$ is computed as follows:

$$P(w_1, \dots, w_m) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-2}, w_{i-1}) \tag{2.3}$$

The trigram model, which considers the past two words of a sequence, can be generalized to the N-gram model, which considers the past $N-1$ words [30]:

$$P(w_1, \dots, w_m) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-N+1}, \dots, w_{i-1}) \tag{2.4}$$

The underlying assumption that the probability of a word depends only on a limited number of previous words is called *Markov assumption*. A simple approach to computing trigram or N-gram probabilities is *maximum likelihood estimation* (MLE) [30]. The MLE estimate for the N-gram probability of a word w_i given a previous sequence of words $h = w_{i-N+1}, \dots, w_{i-1}$ can be computed by counting how many times w_i appeared in the context h, and normalizing by all observations of h [19, 30, 43]:

$$P(w_i \mid w_{i-N+1}, \dots, w_{i-1}) = \frac{\text{count}(w_{i-N+1}, \dots, w_{i-1}, w_i)}{\text{count}(w_{i-N+1}, \dots, w_{i-1})} \tag{2.5}$$

For example, consider that the context h is given by the words "standing on" and we want to predict the probability that the next word w is "the". Given a training corpus, a trigram model would count how many times "standing on" was followed by "the" and compute:

$$P(w_i \mid w_{i-2}, w_{i-1}) = P(\text{the} \mid \text{standing on}) = \frac{\text{count}(\text{standing on the})}{\text{count}(\text{standing on})} \tag{2.6}$$

A problem of N-gram models is that even with very large corpora, computing N-Gram probabilities is difficult. Many word sequences tend to occur only rarely or not at all [19, 43, 55]. Consider the sequence "three guys standing". What is $P(\textit{standing} \mid \textit{three guys})$? A training corpus might not contain any instance of the sequence. Consequently, *count(three guys standing)* would be zero and hence, $P(\textit{standing} \mid$

three guys) would be zero. However, the sequence "three guys" might still occur in the corpus several times. Predicting

$$P(\textit{standing} \mid \textit{three guys}) = 0$$

would underestimate the true probability of the sequence.

Using a standard N-gram model would result in many such zero probabilities. Consequently, the model's predictions would be very noisy. To circumvent probabilities of zero, *smoothing* techniques need to be applied. Smoothing takes away some probability mass from frequent events and redistributes it to unseen events (i. e. those of zero probability) [19, 30, 43]. Several smoothing techniques exist, for example *interpolated Kneser-Ney smoothing* [6, 32]. A detailed introduction to interpolated Kneser-Ney smoothing and other smoothing methods for NLP can be found in Jurafsky and Martin [30].

2.3 WORD EMBEDDINGS

The idea of representing words as continuous vectors has been studied extensively in the past [2, 4, 35, 45, 49, 68]. Several models for learning word representations (word vectors) exist. An introduction and comparison of the most popular models can be found in Lai et al. [35]. Vector-space representations of words are important for various NLP tasks and have improved their performance significantly [8, 9, 43, 53, 61]. As outlined by Luong et al. [38]:

> *"The use of word representations [...] has become a key 'secret sauce' for the success of many NLP systems in recent years, across tasks including named entity recognition, part-of-speech tagging, parsing, and semantic role labeling."*

The basic idea of the approach is to represent words as compact and dense[2] d-dimensional vectors of real numbers, such that each number in the representation defines some distinct informative property. Such distributed word representations[3] are often based on the *distributional hypothesis* which states that words occurring in a similar context tend to have similar meanings [24].

The number of vector dimensions d depends on the task and corpus size and tends to lie between 50 and 500 [30]. The word vectors are also called *word embeddings* because each word is embedded in a vector space. The term *word embedding* was established by Bengio et al. [2] in 2003 who trained a neural language model to simultaneously learn distributed representations of words and the probability for

2 Dense refers to the characteristic that most values of the vector are non-zero.

3 When using distributed representations, each item is represented using several representational elements, e. g. using a multi-dimensional vector. Instead of representing an entire item, each element in the representation denotes a *feature* that can be associated with more than one item.

word sequences. However, models for learning vector-space representations of words have already been studied much earlier in the context of *information retrieval*. Well known earlier models include Latent Semantic Analysis (LSA) [11] or Latent Dirichlet Allocation (LDA) [3]. The widespread use and popularity of word embeddings present today can be attributed to Mikolov et al. [46] who created *word2vec*, a technique described in more detail in section 2.3.1.

Using vector-space representations of words has several advantages compared to representing them as indices in a vocabulary. While the latter approach is simple and robust, the resulting representations have no natural notion of similarity, i. e. they do not capture similarities and relationships between words. In contrast, distributed representations allow words with similar meanings or grammatical properties to have similar vectors [30]. Also, it was found that word embeddings encode many linguistic patterns and regularities which can often be represented as linear translations. For example, as outlined in Mikolov et al. [45]: "the result of the vector calculation vec("Madrid") - vec("Spain") + vec("France") is closer to vec("Paris") than to any other word vector".

In the following we will describe different word embedding models applied in this work.

2.3.1 *Word2Vec*

The most popular technique for learning word embeddings is *word2vec*, which was introduced by Mikolov et al. [46]. Word2vec is a family of two related models - the Skip-gram and the CBOW model. The major difference between the models is that the CBOW architecture predicts a target word based on surrounding context words, while the Skip-gram predicts surrounding words given a target word. Both architectures are shallow, two-layer neural networks that are trained on a large number of text documents. In the following, we will focus on the Skip-gram model as an example for the functioning of word2vec. The basic architecture of the model is illustrated in figure 2.1

As mentioned, the Skip-gram model is trained to predict context words given a target word. Context words are words within a specified window size around the target word. For example, consider the following sentence and the target word "guys":

all of a sudden I noticed three guys standing on the sidewalk

Given a window size of two, context words would be "noticed", "three", "standing" and "on". During training, the Skip-gram network is given word pairs of *(target, context)* words found in the training documents. In our example, possible word pairs would be *(guys, noticed)*, *(guys, three)*, *(guys, standing)* and *(guys, on)*. Each word is represented as a one-hot vector, with as many components as words in the vocabulary. For example, with a vocabulary of 1000 words, "guys" would be represented by a 1000-dimensional vector, with a single entry of 1 at the position corresponding to the word "guys" and 0's at all other positions.

The output of the Skip-gram model is given by a single vector, also with as many components as words in the vocabulary. Each element in the output vector represents the probability of the corresponding word being a randomly selected context word of the input word.

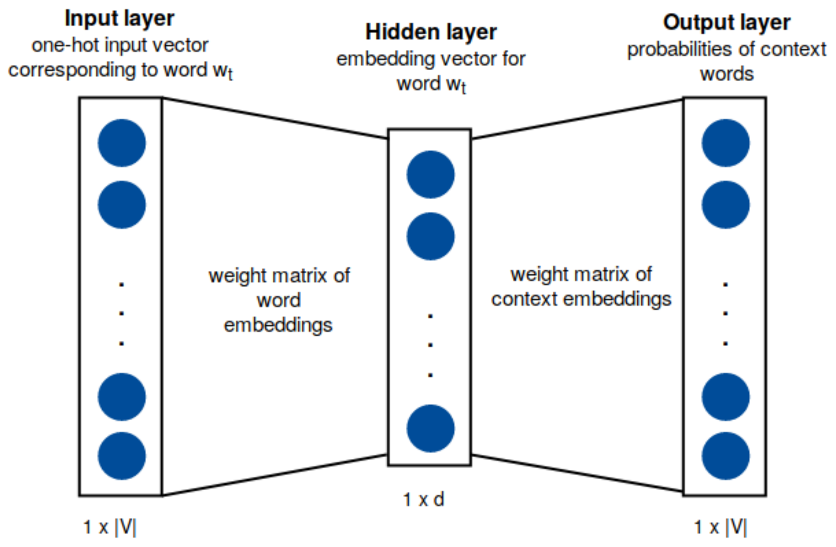

Input layer
one-hot input vector corresponding to word w_t
Hidden layer
embedding vector for word w_t
Output layer
probabilities of context words
weight matrix of word embeddings
weight matrix of context embeddings
1 x |V|
1 x d
1 x |V|

Figure 2.1: Architecture of the Skip-gram model

The Skip-gram's training objective is to learn word representations that are good at predicting context words. More formally, given a sequence of T training words $w_1, \ldots, w_T$, the objective function of the model which is maximized is given by [45]

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leqslant j\leqslant c, j\neq 0}\log p(w_{t+j} \mid w_t) \tag{2.7}$$

where w_t is the target (or center) word, and c is the size of the context window. In the basic Skip-gram, the probability $p(w_{t+j} \mid w_t)$ is defined using the softmax function:

$$p(w_o \mid w_c) = \frac{\exp(u_{w_o}^T v_{w_c})}{\sum_{w=1}^{W}\exp(u_w^T v_{w_c})} \tag{2.8}$$

where w_o represents the outside (or context) word, w_c the center (or target) word, u_{w_o} and v_{w_c} the corresponding word vectors, and W the number of words in the vocabulary. The most important part of this equation is the dot product between the vector representing the context word and the vector representing the target word. The dot product between two vectors can be used to measure their similarity, because the resulting value will be high when the vectors have large values in the same dimensions [30]. The softmax function normalizes the dot product into a probability.

Problematic about this formulation of $p(w_o \mid w_c)$ is the computational complexity of computing the denominator, which requires computing the dot products between the embedding of the target word and the embeddings of all other words in the vocabulary. Two alternative solutions were proposed [45]. The first one is the hierarchical softmax, a computationally efficient approximation to the full softmax. The second alternative is Negative sampling (NEG), a simplification of Noise Contrastive Estimation (NCE). When training the Skip-gram model with negative sampling, for each target word, the algorithm will choose the surrounding context words as positive examples, and for each positive example a number of k negative (or noise) examples (non-neighboring words). The goal is to learn embeddings close to the context words and distant from the noise words [30]. In addition to these methods, higher-frequency words in the corpus are usually downsampled. This strategy is employed to counter the imbalance between rare and frequent words which is present in most training corpora. Details on these techniques can be found in Mikolov et al. [45].

The Skip-gram model is trained using gradient ascent (see section 2.4.4) to maximize the objective function given in equation (2.7). After training, the model has learned two separate embeddings for each word - a *word embedding*, given by the rows of the weight matrix between input and hidden units, and a *context embedding*, given by the columns of the weight matrix between hidden and output units. These word embeddings are typically averaged or concatenated to form a single word embedding per word [60].

Although the CBOW model is algorithmically similar to the Skip-Gram model, their differences often cause one of them to be superior for a particular task. For example, Chiu et al. [7] reported that Skip-gram vectors show better results in word similarity tasks and entity mention tagging compared to CBOW.

2.4 NEURAL LANGUAGE MODELS

The idea of modeling language with neural networks has been subject to research for many years. The first large-scale neural language model was proposed by Bengio et al. [2] in 2003 and based on a simple feedforward neural network. But only the introduction of Recurrent Neural Network Language Models (RNNLMs) by Mikolov et al. [44] in 2010 established neural networks as state-of-the-art for language modeling, replacing standard N-gram techniques. The model proposed in this work is based on a Recurrent Neural Network (RNN).

In the following, we will introduce the basic architecture and functioning of artificial neurons, feedforward neural networks, RNNs and other concepts employed in our work.

2.4.1 *Artificial Neurons*

An artificial neuron is the building block of an artificial neural network and elementary for its functioning. Compared to biological neurons it is a very simple abstraction and rather primitive.[4]

An artificial neuron is made up of several elements. The d input signals of a neuron form a vector $\boldsymbol{x} \in \mathbb{R}^d$. Each input $x^{(i)}$ is multiplied by a synaptic weight $w^{(kj)}$ where k refers to the neuron in question and j to the (respective) input neuron. The synaptic weights represent the connections between different neurons. Hence, the output of one neuron is the input to another neuron. The value of the synaptic weight connecting two neurons can be positive or negative, determining how much influence the output of the first neuron has on the second connected neuron.

In addition to the synaptic weights, an artificial neuron is subject to a bias $b^{(k)}$. The bias acts as a threshold by either increasing (if the bias is positive) or decreasing (if the bias is negative) the activation of the neuron. The sum of the weighted inputs and the bias constitutes the activation $s^{(k)}$ of the neuron. In mathematical terms, this can be expressed by the following equation:

$$s^{(k)} = b^{(k)} + \sum_{j=1}^{d} w^{(kj)} x^{(j)} \tag{2.9}$$

To simplify notation, the bias $b^{(k)}$ can be included in the sum by adding a constant component of $x^{(0)} = 1$ to the input vector and a weight $w^{(k0)} = b^{(k)}$ to the weight vector $\boldsymbol{w}$. Equation 2.9 can then be reformulated as:

$$\begin{aligned} s^{(k)} &= \sum_{j=0}^{d} w^{(kj)} x^{(j)} \\ &= \boldsymbol{w}^{\mathsf{T}} \boldsymbol{x} \end{aligned} \tag{2.10}$$

The output $\hat{y}^{(k)}$ of a neuron is computed by applying an activation function σ to the activation $s^{(k)}$:

$$\hat{y}^{(k)} = \sigma(s^{(k)}) \tag{2.11}$$

The activation function is also known as a *squashing function*, since it limits the possible range of the neuron's output. Several choices of

4 This entire chapter is adapted from my Bachelor thesis [50]

activation functions exist, for example the logistic or rectifier function. An illustration of a simple formal neuron with the above mentioned properties can be found in figure 2.2.

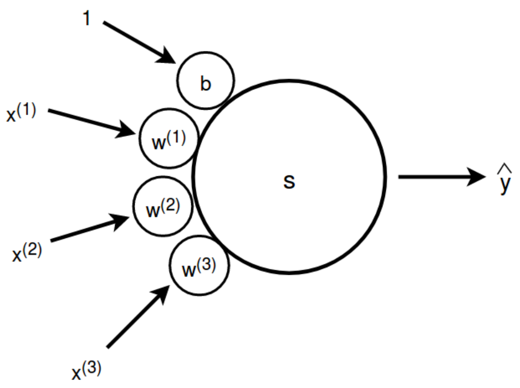


Figure 2.2: A simple model of a formal neuron with weights $\boldsymbol{w} = [b, w^{(1)}, w^{(2)}, w^{(3)}]$. The neuron is subject to an input vector $\boldsymbol{x} = [+1, x^{(1)}, x^{(2)}, x^{(3)}]$. The output $\hat{y}$ of the neuron is computed by applying an activation function σ to its activation s.

2.4.2 *Feedforward Neural Networks*

Feedforward neural networks are a class of machine learning models that try to approximate some function f^* by defining a mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ for an input $\boldsymbol{x}$ and learning the value of the parameters $\boldsymbol{\theta}$ that yield the best function approximation [18].[5] The models are called feedforward because the flow of information from the input $\boldsymbol{x}$ to the output $\boldsymbol{y}$ goes into only one direction. To be more precise: in a feedforward network, no recurrent connections feeding outputs of the model back into itself exist [18].

Feedforward neural networks are called networks because they contain multiple layers of artificial neurons. The most basic architecture consists of an input layer, one or more intermediate (also called hidden) layers and an output layer. The subsequent layers represent a composition of functions.

For example, with an input layer parametrized by $\boldsymbol{W}^{(1)}$ and representing some function $f^{(1)}$, one hidden layer parametrized by $\boldsymbol{W}^{(2)}$ and representing some function $f^{(2)}$ and an output layer, parametrized by $\boldsymbol{W}^{(3)}$ and representing some function $f^{(3)}$, the network mapping would be given by $f(\boldsymbol{x}) = f^{(3)}\big(f^{(2)}(f^{(1)}(\boldsymbol{x}))\big)$. As explained in section 2.4.1 the output of a single artificial neuron is given by $\sigma\big(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}\big)$. For a whole layer of artificial neurons the output is given by the vector $\sigma\big(\boldsymbol{W}^\mathsf{T}\boldsymbol{x}\big)$ that contains the output of each neuron in the layer. $\boldsymbol{W}$ represents the weight matrix containing the weight vectors and biases of all neurons, and the activation function σ is applied element-wise.

5 For simplicity, we will leave out $\boldsymbol{\theta}$ and write only $f(\boldsymbol{x})$ in the following.

With this knowledge the mapping represented by the whole network can be rewritten as:

$$\begin{aligned} f(\boldsymbol{x}) &= f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x}))) \\ &= \underbrace{\sigma^{(3)}\left(\boldsymbol{W}^{(3)}\underbrace{\left(\sigma^{(2)}\left(\boldsymbol{W}^{(2)}\left(\underbrace{\sigma^{(1)}(\boldsymbol{W}^{(1)}\boldsymbol{x})}_{\text{output layer 1}}\right)\right)\right)}_{\text{output layer 2}}\right)}_{\text{output layer 3}} \end{aligned}$$

During training, $f(\boldsymbol{x})$ is driven to match the target mapping $f^*(\boldsymbol{x})$ [18]. A more detailed introduction to feedforward neural networks can be found in Goodfellow et al. [18].

2.4.3 *Recurrent Neural Networks*

An RNN is a special type of neural network for processing sequential data. It differs from a feedforward neural network by containing recurrent (cyclic) connections. Many different architectures for RNNs exist. As pointed out by Goodfellow et al. [18]: "Much as almost any function can be considered a feedforward neural network, essentially any function involving recurrence can be considered a Recurrent Neural Network".

A central difference between feedforward and recurrent networks lies in the way in which parameters are shared between different parts of the model. Parameter sharing allows a model to be extended to examples of different lengths and to generalize across examples [18]. As pointed out by Goodfellow et al. [18] parameter sharing is especially important when the same piece of information occurs at several positions within an input sequence. As an example, the authors name the two sentences "I went to Nepal in 2009" and "In 2009, I went to Nepal". When training a machine learning model to extract the year of travel, we would like it to recognize the year 2009 independent of whether it appears at the sixth or second position in the sentence. A feedforward network that processes sentences of fixed length would have different parameters for each input feature. Therefore, it would have to separately learn all the rules of the language at each sentence position. Different to a feedforward network, an RNN would share the same weights across multiple time steps [18].

The sharing of parameters arises from the way in which an RNN operates: when computing the output of a hidden unit, each component of the output is produced by applying the same update rule to each component of the previous output. When using a basic RNN the values of the hidden units $\boldsymbol{h}$ at time step t can be described as follows [18]:

$$\boldsymbol{h}^{(t)} = g(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}, \boldsymbol{\theta}) \tag{2.12}$$

with g being the update rule, $\boldsymbol{h}^{(t-1)}$ being the value of the hidden unit at time step $t-1$, $\boldsymbol{x}^{(t)}$ being the input vector at time step t and $\boldsymbol{\theta}$ being the parameters of the model. Furthermore, the model typically contains an output layer that uses information from the hidden state in order to make predictions. Because the state of a hidden neuron at time step t is a function of all inputs from previous time steps, the recurrent connections can be viewed as creating a kind of "memory". A hidden unit that preserves information across multiple time steps is called a *memory cell* or *cell* [15, 18, 20]. The exact form of the update rule g depends on the type of recurrent unit. In the following, we will describe a basic type of cell for which the transition from one hidden state to the next is based on an affine transformation followed by a point-wise nonlinearity. Several other types of cells exist. One of them, Long Short-Term Memory (LSTM) cells, are discussed in chapter 2.4.7.

An example of a basic RNN is illustrated in figure 2.3. The figure shows the computational graph of an RNN that maps an input sequence $\boldsymbol{x}$ to an output sequence $\boldsymbol{o}$. The loss L measures how far each output $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$. For example, when using a softmax output layer, the outputs $\boldsymbol{o}$ are considered to be unnormalized log probabilities. The loss then internally computes $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{o})$ to receive normalized probabilities and compares these values to the targets $\boldsymbol{y}$. The input-to-hidden connections of the network are parametrized by a weight matrix $\boldsymbol{U}$, the hidden-to-hidden connections are parametrized by a weight matrix $\boldsymbol{W}$ and the hidden-to-output connections are parametrized by a weight matrix $\boldsymbol{V}$ [18].

For an input vector sequence $\boldsymbol{x}^{(1)}, \dots \boldsymbol{x}^{(\tau)}$ and an initial hidden state $\boldsymbol{h}^{(0)}$ the activations of the hidden units for $t = 1$ to $t = \tau$ are computed as follows [18, 20]:

$$\boldsymbol{h}^{(t)} = \sigma_h(\boldsymbol{b} + \boldsymbol{W}\,\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\,\boldsymbol{x}^{(t)}) \tag{2.13}$$

where $\boldsymbol{b}$ denotes the bias and σ_h the element-wise activation function in the hidden layer. Given the values of the hidden units, the output is computed using the following equation:

$$\hat{\boldsymbol{y}}^{(t)} = \sigma_y(\boldsymbol{c} + \boldsymbol{V}\,\boldsymbol{h}^{(t)}) \tag{2.14}$$

where c denotes the bias and σ_y the element-wise activation function of the output layer (e. g. softmax).

The loss for a sequence of $\boldsymbol{x}$ values and corresponding targets $\boldsymbol{y}$ is given by the sum of the individual losses. If, for example, the loss

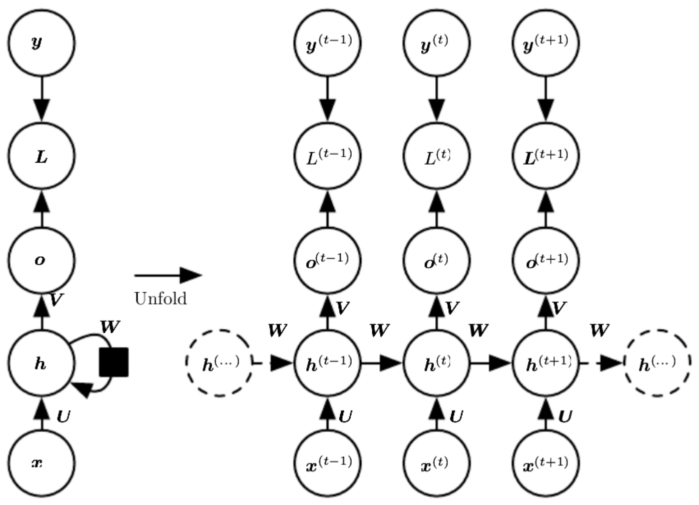


Figure 2.3: Computational graph of an RNN that maps an input sequence $\boldsymbol{x}$ to an output sequence $\boldsymbol{o}$. The loss L measures how far each output $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$. (Left) RNN and loss with recurrent connections. (Right) RNN and loss as an time-unfolded computational graph. Taken from Goodfellow et al. [18]

is given by the negative log-likelihood of $\boldsymbol{y}^{(t)}$ given $\boldsymbol{x}^{(1)}, \dots \boldsymbol{x}^{(t)}$ we have [18]:

$$\begin{aligned} L(\boldsymbol{x}^{(1)}, \dots \boldsymbol{x}^{(\tau)}, \boldsymbol{y}^{(1)}, \dots \boldsymbol{y}^{(\tau)}) &= \sum_t \underbrace{-\log p_{\text{model}}(y^{(t)} \mid \boldsymbol{x}^{(1)}, \dots \boldsymbol{x}^{(\tau)})}_{:= L^{(t)}} \\ &= \sum_t L^{(t)} \end{aligned} \tag{2.15}$$

where $\log p_{\text{model}}(y^{(t)} \mid \boldsymbol{x}^{(1)}, \dots \boldsymbol{x}^{(\tau)})$ is given by selecting the entry for $y^{(t)}$ from the model's output vector $\hat{\boldsymbol{y}}^{(t)}$ and where $L^{(t)}$ represents the per-example loss function [18].

The partial derivatives of the loss with respect to the parameters can be computed efficiently by applying Backpropagation Through Time (BPTT) as explained in section 2.4.4. Afterwards, the network can be trained with Stochastic Gradient Descent (SGD). Recurrent Neural Networks are universal approximators to the effect that any function computable by a Turing machine can be computed by some RNN of finite size [18].

One major challenge for deep RNNs is to learn long-term dependencies. In simple terms, the problem is that gradients propagated over many stages tend to either explode or vanish [1, 26]. Various approaches to solve this difficulty exist. For example, so called "skip connections" can be added to the network which directly connect variables from the distant past to variables in the present [36]. An approach that has been proven to be extremely effective in the past [20, 71, 72] is to use gated RNNs, such as the LSTM discussed in section 2.4.7.

2.4.4 *Gradient-Based Training*

As mentioned in section 2.4.3, RNNs can be trained using SGD. Gradient descent is an optimization algorithm that can be used to minimize (or maximize) some function $f(x)$. To minimize f, gradient descent starts with some initial value for x. Then, it updates x in an iterative fashion by making small steps into the direction of the negative gradient, according to some learning rate η:

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \eta \nabla f(\boldsymbol{x}^{(t)}) \tag{2.16}$$

The gradient $\nabla f(\boldsymbol{x}^{(t)})$ points in the direction of the greatest rate of *increase* of f around $\boldsymbol{x}^{(t)}$. Consequently, we can decrease f by moving in the opposite direction. An illustration of the idea behind gradient descent is given in figure 2.4.

In gradient-based training of neural networks f is typically seen as a function of the network parameters and given by the sum over the training examples of some per-example loss function (as, for instance, given in equation (2.15)) [18]. Computing the gradient over all training examples is computationally expensive and slow, especially for large datasets [18]. However, the gradient can be approximated using only a small number of examples. This insight is the basis of stochastic gradient descent.

Stochastic Gradient Descent is an extension of gradient descent in which the gradient is computed using only a mini-batch of training examples. This has several advantages compared to gradient descent. For example, the computational complexity when training on large datasets is much smaller. Also, the system can converge faster. A more detailed introduction and discussion of SGD can be found in Goodfellow et al. [18]. Besides SGD, several other more sophisticated learning algorithms exist. For example, the Adam algorithm [31] automatically adapts the learning rates of each parameter during training. Other algorithms include momentum SGD [64], RMSProp [25] and Adagrad [12].

In order to compute the gradient $\nabla f(\boldsymbol{x}^{(t)})$ needed for SGD, we must determine the partial derivatives of f with respect to each parameter. This can be done using the *backpropagation* algorithm [56]. BPTT is an extension of backpropagation. The main idea behind BPTT is to *unfold* (or unroll) a given recurrent architecture and apply backpropagation to the unfolded computational graph (see right side of figure 2.3) [18, 43].

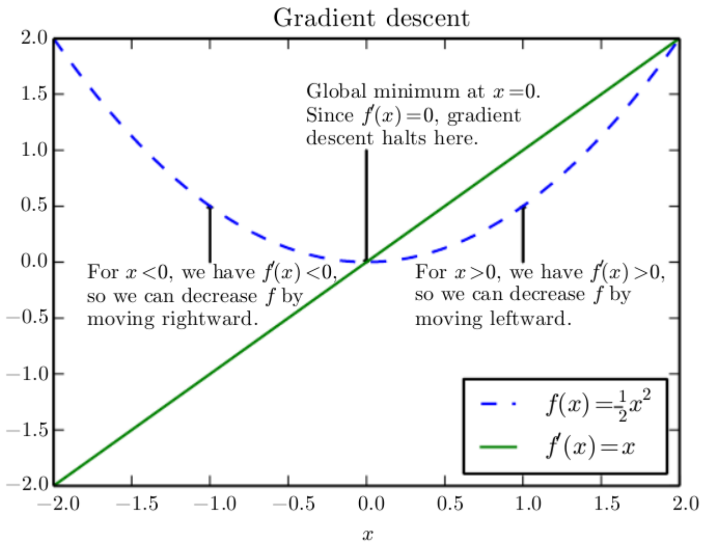


Figure 2.4: An illustration of gradient descent, showing how the derivative of a function can be used to follow it downhill. Taken from Goodfellow et al. [18]

Backpropagation Through Time has several problems. First, a single update step is computationally expensive. For example, computing the gradient of an RNN on sequences with 500 time steps is equivalent to a full forward and backward pass through a feedforward neural network with 500 layers [64]. Second, gradients propagated through many layers tend to either vanish or explode [18, 64]. These problems can be handled by using a further extension called *truncated* BPTT in which the RNN is unfolded only for a fixed number of time steps. Truncated BPTT is the most common method to train RNNs and was first introduced by Elman [13] in 1990. A detailed introduction to truncated BPTT can be found in Mikolov [43] or Sutskever et al. [64].

In the case of neural networks, the objective function f is typically given by some loss function that should be minimized (see e. g. equation (2.15)). A typical loss function for RNNs is the *cross entropy loss* [18]. The cross entropy between the probability distribution $\hat{\boldsymbol{y}}$ predicted by a network and the true distribution $\boldsymbol{y}$ is computed as follows:

$$H_{\boldsymbol{y}}(\hat{\boldsymbol{y}}) = -\sum_i y_i \log(\hat{y}_i) \tag{2.17}$$

During training, the parameters of the network are adjusted using SGD in order to minimize the given loss function. However, minimizing the loss function during training is not the only goal [18]. This will be discussed in more detail in the next section.

2.4.5 *Overfitting and Underfitting*

When training a neural network, minimizing the expected value of the loss function under the data generating distribution is not the foremost goal. Instead, we want our model to accurately capture regularities in the training data (minimizing the training loss) *and* generalize well to new, unseen data (minimizing the so called test loss). [18]. Therefore, minimizing the training loss by simply memorizing the training examples is not desirable. A neural network that achieves a small loss during training, but performs poorly on a separate test set is *overfitting*. Correspondingly, a network that is not able to fit the training data well is *underfitting*.
To monitor whether a network is overfitting, the available data is typically split into a training-, validation- and test set. During training, the network is evaluated on the evaluation set at regular intervals. While the training error decreases steadily with an increasing number of training iterations, the validation error possible starts to increase again at some point. Training should be stopped when the minimum validation error is reached. This strategy is called *early stopping*.

Additionally, several forms of regularization can be used to prevent a network from memorizing the training examples. One popular method is *dropout*, which randomly drops some units and their connections during training. A common and simple way to "drop" a unit from a network is to multiply its output value by zero [18]. When using dropout during training, for each mini-batch, we randomly sample a binary vector (also called binary mask vector) that contains one entry for each input or hidden unit in the network. The entry for each unit is sampled independently from all other entries. The probability of a mask value being one (meaning that the corresponding unit is kept) is a hyperparameter set before training. Typical values are 0.5 for the hidden layers and 0.8 for the input [18]. Each unit is then multiplied by the corresponding mask value causing it to be either included or dropped [18]. A detailed introduction to dropout can be found in Goodfellow et al. [18]. In section 2.4.10 we will introduce several forms of dropout, for example, variational dropout.

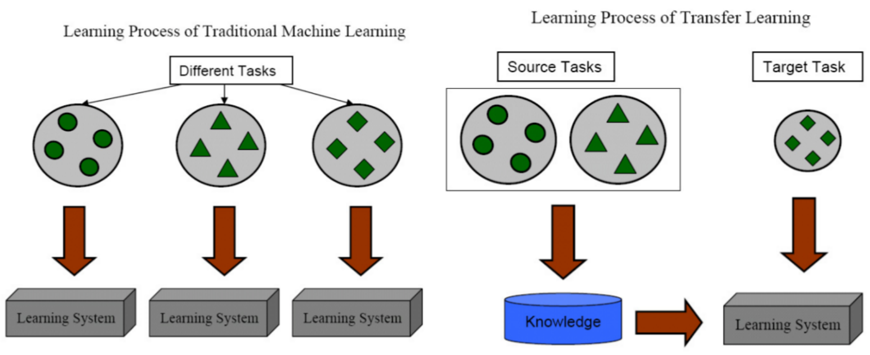
As mentioned already, another challenge for RNNs (and deep neural networks in general) is that derivatives propagated over many time steps tend to be either very large or very small in magnitude. When a parameter gradient is very large, the update performed by gradient descent is likely to throw the parameters into a different region of the objective function, undoing previous optimization steps. This can be prevented using *gradient clipping* [18]. Given the gradient $\boldsymbol{g} = \nabla f(\boldsymbol{x})$, one common form of gradient clipping is to clip the norm $||\boldsymbol{g}||$ of the gradient before performing the parameter update [48]. Given a norm threshold ν, the gradient $\boldsymbol{g}$ is clipped as follows [18]:

$$\text{if } ||\boldsymbol{g}|| > \nu : \boldsymbol{g} \leftarrow \frac{\boldsymbol{g}\nu}{||\boldsymbol{g}||} \tag{2.18}$$

With this adaptation, the parameter update still points into the same direction as the gradient, but its magnitude is bounded. So when the gradient explodes and the gradient descent algorithm proposes to make a large update step, gradient clipping intervenes and avoids performing a harmful step.

2.4.6 *Transfer Learning*

Transfer learning describes the process of extracting knowledge from one or more source tasks to improve the performance on a target task, which is typically related to the source tasks [47]. According to Pan and Yang [47] the motivation behind studying transfer learning lies in the observation that people can intelligently apply already acquired knowledge to solve new problems in a faster or better way. The difference between transfer learning and traditional machine learning is illustrated in figure 2.5 [47]. As illustrated in the figure, given different tasks, traditional machine learning techniques try to learn one model for each task from scratch. Transfer learning, however, tries to apply knowledge acquired on previous tasks to some target task, which typically has fewer high-quality data.



(a) Traditional Machine Learning (b) Transfer Learning

Figure 2.5: Comparison of the learning processes in traditional machine learning and transfer learning. Taken from Pan and Yang [47]

The decision which part of knowledge should be extracted as well as when and how it should be transferred, differs between existing transfer learning techniques. In this work, we focus on the transfer of network weights learned on some dataset to a new network, which is trained on a different but related dataset. This is further described in section 4.

2.4.7 *Long Short-Term Memory*

The LSTM architecture [27] is designed to be better at storing information and finding and learning long-term dependencies than standard recurrent networks [18, 20, 57]. Long Short-Term Memory networks have been extremely successful in a variety of tasks like handwriting recognition [20, 23], speech recognition [21, 22] or machine translation [63].

In many simple RNNs, the activation function of the hidden layer (see equation (2.13)) is given by an element-wise application of some sigmoid function. Gated recurrent networks like LSTMs make use of *gated* activation functions. Long Short-Term Memory architectures contain special *memory cells* (also called state cells) which have an internal recurrence (i. e. a self-feedback loop) and additional connections to different gating units that control the flow of information through the cell [18, 20, 57]. The first gate is called the *input gate*. It controls the flow of inputs into the cell and decides whether or not the current input is worth preserving. Similar in function is the *forget gate* which decides how useful the internal state of the memory cell is before feeding it back into the cell through its self-feedback loop. Thereby, it adaptively forgets and resets the cell's memory. The third and last gate is the *output gate*. It controls which parts of the memory cell state should be present in the hidden state, thereby controlling the output flow into the rest of the network [18, 57]. Additionally, the LSTM architecture can contain *peephole connections* from the memory cell to the multiplicative gates [16]. The structure of a single LSTM cell is illustrated in figure 2.6.

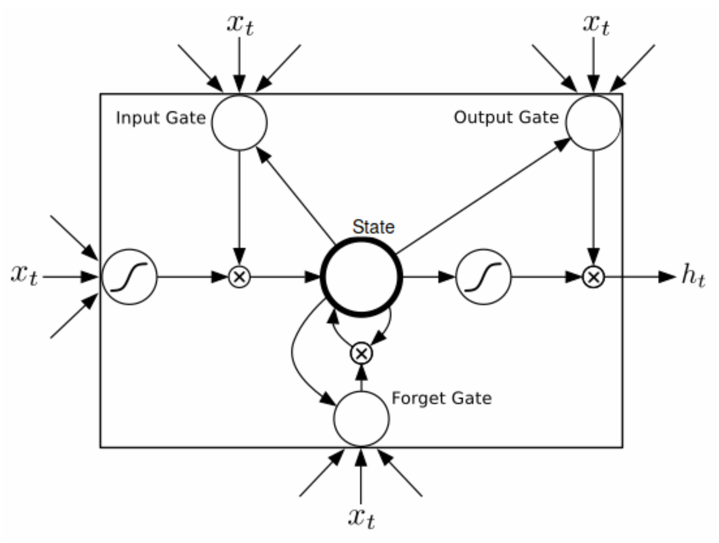


Figure 2.6: Structure of a Long Short-Term Memory Cell adapted from Graves [20]

The LSTM cell used in our models is defined by the following composite function [18]:

$$g_i^{(t)} = \sigma\Big(b_i^g + \sum_j U_{i,j}^g\, x_j^{(t)} + \sum_j W_{i,j}^g\, h_j^{(t-1)}\Big) \tag{2.19a}$$

$$f_i^{(t)} = \sigma\Big(b_i^f + \sum_j U_{i,j}^f\, x_j^{(t)} + \sum_j W_{i,j}^f\, h_j^{(t-1)}\Big) \tag{2.19b}$$

$$s_i^{(t)} = f_i^{(t)}\, s_i^{(t-1)} + g_i^{(t)}\, \sigma\Big(b_i + \sum_j U_{i,j}\, x_j^{(t)} + \sum_j W_{i,j}\, h_j^{(t-1)}\Big) \tag{2.19c}$$

$$q_i^{(t)} = \sigma\Big(b_i^o + \sum_j U_{i,j}^o\, x_j^{(t)} + \sum_j W_{i,j}^o\, h_j^{(t-1)}\Big) \tag{2.19d}$$

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)} \tag{2.19e}$$

where $g_i^{(t)}, f_i^{(t)}, s_i^{(t)}, q_i^{(t)}$, are respectively the *input gate, forget gate, internal state* and the *output gate* for time step t and cell i. The current input vector is given by $\boldsymbol{x}^{(t)}$ and the current hidden vector by $\boldsymbol{h}^{(t)}$. $\boldsymbol{b}, \boldsymbol{U}, \boldsymbol{W}$ are respectively biases, input weights and recurrent weights for the different gates. σ denotes the logistic sigmoid function.

2.4.8 *Recurrent Neural Language Model*

The structure of a language model based on an RNN is similar to the one of a general RNN illustrated in figure 2.3. However, a few important details are different. These will be explained in the following.

The architecture of a basic RNNLM is illustrated in figure 2.7. A model of similar form was employed in this work and has been used in many previous applications (see e. g. [29, 39, 40, 74, 75]). Important to note is that the number of hidden layers typically varies between different publications and that the basic RNNLM architecture can be adapted, for example by using additional inputs [74].

In a word-based RNNLM, the input to the network is given by a sequence of word IDs where each word ID represents a specific word from some vocabulary. The target sequence is given by the input sequence shifted one time step to the left. For example, given the vocabulary *[how, are, were, me, you, today, tomorrow]* and the word-ID mapping *[how: 1, are: 2, were: 3, me: 4, you: 5, today: 6, tomorrow: 7]* the sequence *"how are you"* (with "today" being the next word in the sequence) would be represented by the word ID sequence *[1, 2, 5]*. The target sequence would be given by *[2, 5, 6]*.

Before feeding the input sequence into the hidden units, the word IDs are embedded into a low dimensional vector space, that is, each input word is mapped to its embedding vector using the model's embedding matrix. The embedding matrix is typically initialized randomly and adapted during training [28, 29, 40, 75]. Alternatively,

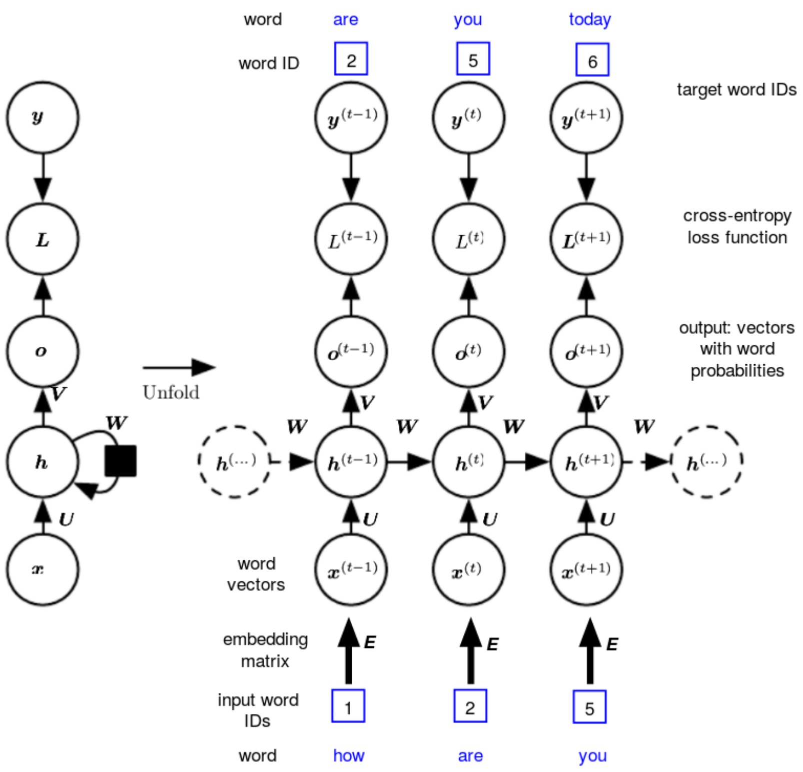


Figure 2.7: Computational graph of an RNNLM that maps an input sequence of word IDs $\boldsymbol{x}$ to an output sequence $\boldsymbol{o}$. The word IDs are embedded into a low dimensional vector space using the model's word embedding matrix $\boldsymbol{E}$. The cross-entropy loss L measures how far each output $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$. Adapted from Goodfellow et al. [18].

it can be initialized with pre-trained word embeddings (see section 4.7.4).

The sequence of word embeddings is processed by one or more layers of LSTM hidden units. The final outputs of the LSTM units are further processed by an output layer with a softmax activation function. For each input word, the output layer produces a vector with as many entries as words in the vocabulary where each entry represents the predicted probability that the corresponding word is the next word in the sequence. The network is trained by minimizing the cross-entropy loss (see equation (2.17)) between the predicted sequence and the target sequence.

2.4.9 *RNNLMs vs. N-grams*

Due to their simplicity, N-gram models were dominant in statistical language modeling for a long time. However, RNNLMs have several advantages compared to N-gram models. For example, as the amount of training data increases, the performance of N-gram models decreases substantially. This is explained in detail in Williams et al. [70]. Also, N-gram models require a lot more memory space compared to RNNLMs. Therefore, they do not scale well with increasing data size [70]. Many other advantages exist. While N-gram models count the occurrences of certain word combinations, RNNs learn a distributed representation for each context. This distribution can be used to obtain a probability distribution over all words [70]. Furthermore, RNNLMs do not need smoothing [30]. More importantly, they can handle longer histories of inputs than N-gram models and are able to incorporate long-range dependencies in their predictions. This is especially important for NLP because dependencies between words often span across several sentences.

2.4.10 *Regularization and Initialization Techniques*

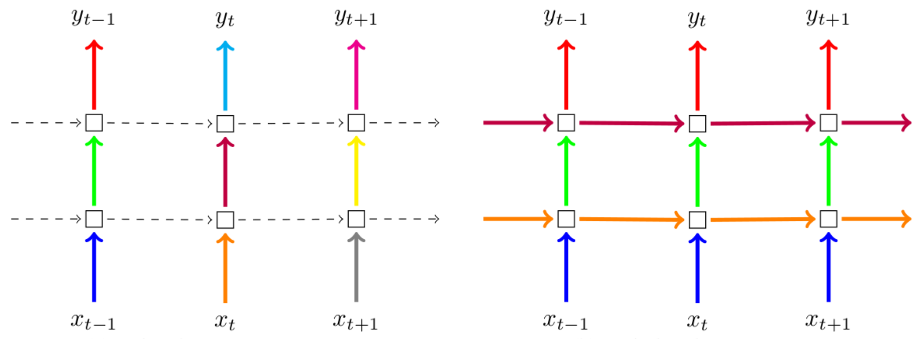
As mentioned in section 2.4.8, the basic architecture of an RNNLM can be adapted in various ways. In our experiments, we tested the effects of several features, which will be introduced in the following sections.

Weight tying

Weight tying [28, 51] refers to sharing the weights between the embedding and softmax layer of a language model, reducing the total number of model parameters substantially. Furthermore, the model does not have to learn a one-to-one correspondence between the input and output. Both consequences of weight tying have been reported to improve LSTM language models [28, 39, 40, 51]. Weight tying works as follows [28]. Given the word embedding matrix $E \in {}^{d_x \times |V|}$, where d_x is the word embedding dimension and $|V|$ the vocabulary size, and given the output projection matrix $V \in {}^{|V| \times d_h}$, where d_h is the size of the RNN hidden state, the output projection matrix is tied to the input embedding matrix by setting $V = E^T$ (see figure 2.7). Furthermore, the output bias vector is set to zero.

Variational dropout

Variational Dropout (VD) is a variant of standard dropout that was introduced by Gal and Ghahramani [14] in 2016. In standard dropout, at each time step, a new binary dropout mask is sampled even if some connection is repeated. For example, the input x_0 to an LSTM at time step $t = 0$ receives a different dropout mask than the input x_1 of the same LSTM at time step $t = 1$. Furthermore, in previous dropout techniques like the one of Zaremba et al. [75] no dropout was applied to the recurrent connections. Different to standard dropout, variational dropout samples a binary dropout mask only once and then repeatedly uses the same mask for all time steps of one training example. Within a mini-batch, each example uses a unique dropout mask. Variational dropout is further illustrated in figure 2.8.



(a) Naive dropout RNN (b) Variational RNN

Figure 2.8: Comparison of naive (standard) dropout (left) and variational dropout (right) taken from Gal and Ghahramani [14]. Each square represents an RNN unit, with horizontal arrows representing time dependence (recurrent connections). Vertical arrows represent the input and output to each RNN unit. Coloured connections represent dropped-out inputs, with different colours corresponding to different dropout masks. Dashed lines correspond to standard connections with no dropout. Current techniques (naive dropout, left) use different masks at different time steps, with no dropout on the recurrent layers. The proposed technique (variational RNN, right) uses the same dropout mask at each time step, including the recurrent layers.

Embedding Dropout

Embedding Dropout (ED), as proposed by Gal and Ghahramani [14], refers to performing dropout on the embedding matrix. When training RNNLMs, the embedding matrix is typically optimised during training and can therefore lead to overfitting. With ED, rows of the embedding matrix are randomly set to zero. This corresponds to randomly dropping specific words throughout the input sequence, encouraging the model's output to be independent of the presence of certain words in the input [14]. Furthermore, since the same dropout mask is used at each time step, the same words are dropped throughout the input sequence. Gal and Ghahramani [14] provide the following example: "The sentence *"the dog and the cat"* might become *"- dog and - cat"* or *"the - and the cat"*, but never *"- dog and the cat"* ".

Pre-trained Word Embeddings

In our experiments we tested the effects of initializing the embedding matrix of our models using pre-trained word embeddings, as opposed to initializing it randomly. We investigated two different word embeddings techniques, which were introduced in chapter 2: the CBOW model and the skip-gram model. We decided not to test the effect of pre-trained GloVe embeddings because of the findings of Cornegruta et al. [10]. Cornegruta et al. investigated potential performance gains of different pre-trained word embeddings when modeling radiological language and found no significant improvements of GloVe embeddings over a random initialization of the embedding matrix.

2.5 EVALUATING LANGUAGE MODELS

The performance of language models can be evaluated in two ways - extrinsically and intrinsically. The two strategies complement each other. Therefore, we evaluated our models using both extrinsic and intrinsic methods.

2.5.1 *Extrinsic Evaluation*

Extrinsic evaluation refers to embedding the language model in an application and measuring how much the application improves [30]. Such an evaluation is the only possibility to determine which modifications will be useful with respect to the desired application. In the case of applying a language model to word prediction, extrinsic evaluation could, for instance, be performed by employing the language model as an autocompletion tool on a smartphone. Users would then judge how good the word predictions are, how often they are used and how much time is saved when using the tool.

A disadvantage of extrinsic evaluation is its computational complexity. Running a big NLP system end-to-end is often expensive and slow to compute. Therefore, it is typically combined with intrinsic evaluation [30].

2.5.2 *Intrinsic Evaluation*

Intrinsic evaluation metrics allow to measure the quality of a model independent of a particular application [30]. For language modeling a typical intrinsic evaluation metric is *perplexity* [19, 30, 43].

The Perplexity (PP) of a language model on a test set $\boldsymbol{w} = \{w_1, \ldots, w_m\}$ is the inverse probability of $\boldsymbol{w}$, normalized by the number of words [30]:

$$\begin{aligned} PP(\boldsymbol{w}) &= P(w_1, \ldots, w_m)^{-\frac{1}{m}} \\ &= \sqrt[m]{\frac{1}{P(w_1, \ldots, w_m)}} \\ &= \sqrt[m]{\prod_{i=1}^{m} \frac{1}{P(w_i \mid w_1, \ldots, w_{i-1})}} \end{aligned} \tag{2.20}$$

As evident in the last line of equation (2.20), the perplexity is low if the conditional probability of the word sequence is high. Therefore, minimizing the perplexity of a test set is equivalent to maximizing the probability of the test set according to the language model [30].

Perplexity has many favourable properties [19, 30, 43]. For example, when given a set of test data, it can be computed easily. Also, it is closely related to the cross entropy (a typical loss function of RNNs) between the model and a test dataset.

The cross entropy, as introduced in equation (2.17) cannot be computed exactly if the true distribution $\boldsymbol{y}$ is not known. This situation occurs in language modeling, where $\boldsymbol{y}$ corresponds to the true distributions of words given the previous words in a corpus, and $\hat{\boldsymbol{y}}$ to the distribution of words predicted by the language model. If $\boldsymbol{y}$ is not known, an approximation to the true cross entropy needs to be computed. Given a model $M = P(w_i \mid w_{i-1}, \ldots, w_{i-N+1})$, the approximation to the cross-entropy of the model on a sequence of words $\boldsymbol{w} = \{w_1, \ldots, w_m\}$ is given by [30]:

$$\begin{aligned} H(\boldsymbol{w}) &= -\frac{1}{m} \log_2 P(w_1, \ldots, w_m) \\ &= -\frac{1}{m} \log_2 \prod_{i=1}^{m} P(w_i \mid w_{i-1}, \ldots, w_{i-N+1}) \end{aligned} \tag{2.21}$$

The relation between this cross-entropy and the perplexity of a model on a sequence of words $\boldsymbol{w}$ is defined as follows [30, 43]:

$$\begin{aligned} PP(\boldsymbol{w}) &= 2^{H(\boldsymbol{w})} \\ &= P(w_1, \ldots, w_N)^{-\frac{1}{m}} \end{aligned} \tag{2.22}$$

3

RELATED WORK

In this section we will discuss research papers related to this thesis. Due to the large amount of existing research we will focus on two topics: first, the application of RNNs to the task of language modeling and second, existing work on improving RNNLMs using transfer learning.

3.1 LANGUAGE MODELS

The task of language modeling has been subject to research for a long time. The beginnings of language modeling research were dominated by simple models, such as N-grams, introduced in section 2.2. The idea of using neural networks for the task of language modeling was first introduced by Elman [13] in 1990 who proposed an RNN for modeling sequences of artificial words. Future work, like the model of Miikkulainen and Dyer [42] proposed in 1991, further developed Elman's architecture to solve higher level natural language processing tasks. The first large-scale and statistical neural language model using real natural language (as opposed to artificial language) was proposed by Bengio et al. [2] in 2003. Although the model of Bengio et al. and later extensions (for example by Schwenk and Gauvain [58]) performed well compared to standard techniques like N-grams, their application was limited because of their high computational complexity. The original model of Bengio et al. needed over three weeks of training on 40 Central Processing Units (CPUs) for only five epochs, despite a small vocabulary (18000 most frequent words of the AP News corpus) and only 60 hidden neurons [2].

Only the introduction of RNNLMs by Mikolov et al. [44] in 2010 established neural networks as state-of-the-art for language modeling, replacing standard N-gram techniques. Mikolov et al. [43, 44] trained RNNs with up to 800 hidden neurons and compared them to state-of-the-art statistical language modeling techniques, showing a significant improvement both in perplexity and word error rate on the Penn Treebank (PTB) dataset, a widely used language modeling benchmark.

Since the publication of Mikolov et al. [44] a lot of further work on RNNLMs has been published. In this context, it is important to distinguish between models performing predictions at the world level, and those performing predictions based on individual characters. Because our work is focused on word-level language models, we will restrict the presented related work to such models.

In 2013, Graves [20] studied the ability of LSTM RNNs to generate complex sequences with long-term structure by predicting one data point at a time. The resulting architectures were evaluated on the PTB dataset and on the Hutter Prize Wikipedia datasets, using both word-based and character-based models. Similar to Graves [20], we use an LSTM architecture and make predictions at the word level. Also, we do not shuffle the sequences during training, and do not reset the state of the LSTM cells during training to allow the network to capture regularities in the data that span many sentences. This approach was adopted in most papers on LSTM language models published after the work of Graves [20].

Zaremba et al. [75] explored the use of dropout for regularizing LSTM networks and validated their findings using language modeling (PTB dataset). Our initial experiments are based on the model architectures tested by the authors and are used as a baseline to study the effects of different architectural features. Further, as proposed by Zaremba et al. we apply dropout in all models. Also, we clip the norm of the gradients to avoid the problem of exploding gradients. Different to the authors, we experiment with variational and embedding dropout and other architectural features that have been proposed in recent papers.

Williams et al. [70] investigated the scaling properties of RNNLMs with respect to model size, training set size, processing power and memory. They trained large recurrent architectures with up to 4096 hidden units and evaluated their performance on different tasks, including language modeling and word prediction. As proposed by Williams et al., we train our architectures on Graphical Processing Units (GPUs). In contrast to Williams et al., we do not train our models with RMSProp, but with SGD. For word-level language models, SGD was found to outperform other methods [40].

Jozefowicz et al. [29] performed an exhaustive study on techniques for extending RNNs for language modeling and tested different architectures on the One Billion Word Benchmark dataset [5]. For example, they studied the effects of varying the number of LSTM layers and the dimensionality of the embedding matrix. Several findings of Jozefowicz et al. are relevant to our work. First, according to the authors, LSTM architectures with a projection layer (i. e. a layer with fewer neurons than the preceding and following layer) trained with truncated BPTT perform well on the task of language modeling. Second, unlike previous work, the authors did not combine their RNNs with N-gram models - an approach adopted in our work. Furthermore, we present data to our models in the same way as the authors. Instead of padding shorter sentences with zeros, a new sentence starts as soon as the previous sentence ends. This maximizes the occupancy per batch and allows networks to learn transitions between sentences.

Press and Wolf [51] and Inan, Khosravi, and Socher [28] concurrently investigated the effects of tying together the input embedding matrix and the output projection matrix of language models. While Press and Wolf [51] provided purely empirical work, Inan, Khosravi, and Socher [28] presented a theoretical justification for this approach. Both papers evaluated the effects of weight tying using multiple datasets and reported improvements in perplexity, while greatly reducing the number of model parameters to be trained. For this reason, we employ weight tying in our models.

Melis et al. [39] reevaluated different regularization methods and architectures using automatic black-box hyperparameter tuning. This strategy was used to eliminate usual sources of experimental variation in order to allow for a qualitative comparison between the methods. In their work, the authors focused on LSTM models and Recurrent Highway Networks [76] and tested the effects of tuning various hyperparameters, including input embedding dropout, variational dropout and output dropout. Their findings suggest that the performance of language models depends strongly on their hyperparameter values and that well-regularised and tuned LSTM models outperform the more recent Recurrent Highway Networks. Like Melis et al., we employ and evaluate weight tying, embedding dropout and variational dropout in our models. Our observations on the PTB dataset validate the findings of Melis et al. who reported that untying input and output embeddings and removing embedding dropout worsens perplexity. Further, inspired by the findings of the authors, we use random search to optimize the hyperparameters of our models.
Similar to Melis et al. [39], Merity et al. [40] studied different strategies for regularizing and optimizing language models based on LSTM units. In accordance with Melis et al. [39], the authors reported that LSTM architectures can outperform more complex architectures when using well tuned hyperparameters. In their work, the authors investigate the effects of several regularization techniques, including variational dropout, embedding dropout, weight tying and independent embedding and hidden sizes. As proposed by Merity et al., we train our models with stochastic gradient descent. For the task of neural language modeling, stochastic gradient descent was found to outperform more advanced methods such as Adam or RMSProp [40]. Furthermore, inspired by the findings of Merity et al., we use independent embedding and hidden sizes. Merity et al. reported that matching the sizes of the embedding vector and hidden layers causes both a degradation in performance and a substantial increase in the total number of parameters.

3.2 TRANSFER LEARNING IN RECURRENT LANGUAGE MODELS

To improve the performance of RNNLMs, limited work has been directed towards the application of transfer learning in recurrent language models.

Tseng, Lee, and Lee [67] investigated how RNNLMs can be turned into personalized language models. In their work, the authors trained a general language model on a single corpus and transformed it into a personalized language model by augmenting the training input with an additional feature vector. The additional feature vector encoded user specific characteristics such as information about the topics talked about by the user. Similar to Tseng, Lee, and Lee [67], we try to improve general RNNLMs by conditioning them on a specific topic. However, our work is not focused on user-specific language models but on topic-specific language models. Also, we do not augment the training examples with additional inputs but transfer weights learned on a general corpus to a new language model that is trained on topic-specific data.

The use of transfer learning for creating personalized language models was further studied by Yoon et al. [74] in 2017. In their work, the authors trained a base LSTM RNN language model on a large dataset and transferred the learned weights to target language models which were fine-tuned with a small amount of user data. Two types of target language models were trained: a sentence completion language model and a message-reply prediction language model. For both models, three transfer learning schemes were investigated, including re-training of the entire architecture with user-specific data and fine-tuning only single layers. While all schemes yielded improvements in perplexity, the scheme that inserted and fine-tuned only one surplus layer was reported to be most successful. The scheme worked as follows: after training a model with general data, a surplus layer was inserted between the output layer and the last LSTM layer. Then, only the parameters of the surplus layer were updated with private user data in the transfer learning phase.

Our work is similar to Yoon et al. in that we first train a general language model and transfer learned weights to a new model which is fine-tuned using topic-specific data. Different to Yoon et al., we do not focus on creating user-specific language models. The transfer learning techniques studied by Yoon et al. were concentrated on creating personalized language models with a small amount of user data and limited computing resources. This setting applies to the mobile device environment in which private user-specific data is located on the mobile phone and should not be transferred out of the device. In contrast to Yoon et al., our work is focused on creating topic-specific language models. In addition, the application of transfer learning is not limited by the amount of computing resources.

While the work on transfer learning for RNNLMs is limited, transfer learning has been applied successfully to other NLP tasks. For example, Zoph et al. [77] proposed a transfer learning method for improving the performance of encoder-decoder networks for neural machine translation. The key idea of their approach is similar to our work: a parent model is trained and some of its parameters are transferred to a second, child model. Transfer learning has further been applied successfully to sentiment analysis, document classification and speech recognition [69].

4

EXPERIMENTS

4.1 INITIAL EXPERIMENTS - PENN TREEBANK

We performed initial experiments using the PTB dataset. PTB has been used for benchmarking language models for many years. It consists of 929k training words, 73k validation words, and 82k test words. We used the pre-processed version introduced by Mikolov et al. [44]. As part of the pre-processing by Mikolov et al. [44], all words were lower-cased, newlines were replaced with <*eos*> (end-of-sentence) and all other punctuation, as well as numbers, were removed. The vocabulary was restricted to the 10000 most frequent words, all other words were replaced with an <*unk*> (unknown) token. This vocabulary size is quite small compared to modern datasets like the Wikitext-2 dataset [41] with a vocabulary of 33278 words. Nevertheless, PTB is still widely used in experiments (see e. g. [28, 40, 73]).

4.1.1 *Model and Training Details*

We first reproduced the results of Zaremba et al. [75], using a 2-layer LSTM architecture with the same number of neurons in each layer. We tested three different network sizes: large (1500 units), medium (650 units) and small (200 units). For the large and medium model we closely followed the hyperparameter settings of Zaremba et al. [75]. All models were trained with SGD, using an initial learning rate of 1.0 and decaying it after a certain number of epochs. The large model was trained for 14 epochs with a learning rate of 1.0. After that, the learning rate was reduced with a decay rate of 1/1.15 per epoch until a limit of 55 epochs had been reached. The medium model was trained for 39 epochs, decaying the learning rate with a factor of 1/1.2 after 6 epochs. The small model was trained for 25 epochs, decaying the learning rate with a factor of 1/1.25 after one epoch. The norm of the gradients was clipped at 10 for the large model and at 5 for the medium and small models. All models were trained with a batch size of 20 and unrolled for 35 steps for backpropagation. In the baseline version we used dropout as proposed in Zaremba et al. [75]. Dropout rates were 65% for the large network, 50% for the medium network and 30% for the small network. Further details on the large and medium network can be found in Zaremba et al. [75].

4.1.2 *Results*

To investigate the effects of different architectural features and to check the correctness of our implementation, we trained several models. Models using standard dropout as proposed in Zaremba et al. [75] were used as a baseline and are referred to as LSTM-SD. We trained one baseline model for each network size (small, medium and large) and several other models of small and medium size. All models were evaluated using the average per-word perplexity on the test set. Table 4.1 lists a comparison of the best performing models and the baseline models. Detailed tables including all trained models can be found in the appendix (see section A.1).

Network	**Model**	**Valid**	**Test**	Valid	Test
				Zaremba et al.	
Small	LSTM-SD	99.33	96.14	-	-
(200 units)	LSTM-SD+TW+ED	95.7	91.79	-	-
Medium	LSTM-SD	86.11	82.88	86.2	82.7
(650 units)	LSTM-VD+TW+ED	82.0	78.64	-	-
Large	LSTM-SD	82.4	79.17	82.2	78.4
(1500 units)					

Table 4.1: Comparison of the word-level perplexities on the validation and test set for different models tested on the PTB dataset. VD stands for variational dropout, TW for weight tying, ED for embedding dropout.

When comparing the performance of the different models, we observed the following effects:

1. Using ED dropout in addition to standard or VD improved the results only in some cases. For example, when using an input and output dropout probability of 0.3 in the small model, additional ED of 0.3 worsened the test perplexity by about 6 points. However, when only using dropout probabilities of 0.1, it improved the perplexity by about 5 points.

2. The same applies to weight tying. For example, when using VD with dropout probabilities of 0.2 (small model), additional weight tying worsened the test perplexity by approximately 3 points. However, with dropout probabilities of 0.1, additional weight tying improved the perplexity by about 3 points.

3. Also variational dropout improved the results only for some models. When comparing small models using standard dropout

(applied to the input and output of the LSTM) with small models using VD (applied to the input, recurrent connections and output), the latter improved the model's performance only in some cases. For example, the model with standard dropout of 0.3 performed better than the model with VD of 0.3. However, the model with standard dropout of 0.1 performed worse than the model with VD of 0.1.

4. When using VD instead of standard dropout in the medium models, we sometimes observed extremely high perplexities (> 600). These effects vanished when tying the embedding and softmax weights.

4.1.3 *Discussion*

We believe that the addition of ED and weight tying when already using standard dropout improves the performance if the overall amount of regularization is not too high. This is supported by our observations and holds both for the small and medium models. When tying the weights between the embedding and softmax layer, the total number of model parameters is reduced. Therefore, when applying weight tying in addition to dropout with a high dropout probability, the model performance might be worse due to a reduced capacity to fit the data.

A similar argument holds for the application of VD in small models (see observation 3). When using VD, not only the input and output connections of the recurrent units are regularized but also the recurrent connections. This additional amount of dropout can have a negative effect if combined with other regularization techniques. This could be explained by the fact that the overall regularization gets too high. However, we could not find an explanation for the high perplexity values observed for the training of some medium size models with VD.

Although the best medium model employs VD (test perplexity of 78.64), we could not find VD to be superior to standard dropout in general. The best medium model with standard dropout achieved a test perplexity of 79.04 which is very close to the model with VD. Such a small difference in perplexity might simply be caused by the random initialization. Also, we did not perform extensive hyperparameter tuning. Further testing might refine the observed effects.

4.2 DATASETS AND PREPROCESSING

We trained our models on two different German datasets. The first dataset consists of radiological reports written by clinicians from Essen University Hospital. All patient-identifying data, such as names, addresses or dates of birth were removed by the hospital.

The second dataset includes forum posts from four different forums of *motortalk*, a big German-language forum on all kinds of motorized vehicles[1]. Posts from the forums (and sub forums) of the following makes of car were included in the dataset: Mercedes, VW, BMW, Opel.

The data preparation process had the following steps:

- All punctuation was removed.
- All duplicated words were removed. We did not remove stop words from the corpora because they should be included in the final model predictions.[2]
- All strings containing numbers were mapped to a special *number* string.
- All reports were converted to lower case.
- Tokenization on word and sentence level was performed using the Python library *spaCy* [78].
- Sentence boundary markers were included at the end of each sentence.
- The size of both datasets was restricted to 3 million words. The final radiological dataset was generated by randomly selecting radiological reports from the corpus. The final motortalk dataset was created by randomly selecting an equal number of sentences from each of the four forums. For each selected sentence, we included ten subsequent sentences as a context. 80% of the data were used for training, the remaining 20% were used for testing. Of the training set, 20% were used as a validation set. This corresponds to 1.920.000 training words, 480.000 validation words and 600.000 test words (about three times the size of the PTB dataset).
- The final vocabulary was constructed by including all words with a count of at least 10. This threshold was chosen to allow learning useful word embeddings of all words.
- Words not included in the vocabulary were mapped to a special *rare* string. The *rare* string and sentence boundary markers were included in the vocabulary.
- The resulting vocabulary sizes were 88575 (motortalk dataset) and 21344 (radiological reports).

1 https://www.motor-talk.de/
2 In NLP stop words refer to extremely common words like and, or, by, from, the, etc.

- The out-of-vocabulary rate was 1.13 % on the motortalk dataset and 0.57 % on the radiological dataset.
- To test the effect of transfer learning we created three more datasets of the same size (3 million words) and with the same split into training, validation and test set. One dataset contained only data from the BMW forums, one dataset contained only reports of patients transferred from the oncological department and one dataset contained only reports for which the radiological examination was given by an X-ray of the patient's thorax while the patient was lying down (abbreviation: KTHL). The vocabulary sizes of these datasets were: 73704 (BMW), 35237 (oncology) and 17657 (KTHL).

4.3 MODEL ARCHITECTURE

In accordance with recent publications [39, 40], we used an LSTM architecture with three hidden layers in all experiments. As pointed out by Merity et al. [40] most early LSTM language models (like the one presented in Zaremba et al. [75]) tied the dimensionality of the word embedding vectors to the dimensionality of the LSTM hidden layer. This increases the total number of model parameters substantially because the embedding and all hidden layers will have the same number of neurons. As further pointed out by Merity et al. [40]: "the easiest reduction in total parameters for a language model is reducing the word vector size". Therefore, like Merity et al., we allowed for the embedding and hidden size to be independent of each other. Consequently, the first and last hidden layers of the LSTM had input- and output dimensionality equal to the reduced embedding size, while the second LSTM layer had a variable number of neurons.

Like Zaremba et al. [75] we initialized the hidden states to zero. During training, we used the final hidden states of each minibatch as the initial hidden state of the subsequent minibatch, that is, we never reset the LSTM states [29, 75]. We used a batch size of 20 and unrolled the network for 35 time steps as done in previous work [28, 39, 75]. All models had a softmax output layer and were given batches of word IDs as an input.

Before feeding the input batches into the LSTM, the word IDs were embedded into a vector space, using the model's embedding matrix. All network weights, including the embedding matrix, were initialized uniformly in the interval $[-0.05, 0.05]$ as done in Zaremba et al. [75]. Because of the positive effects of weight tying reported in the literature [28, 39, 40, 51] and our validation of these findings, we used weight tying in all network architectures. Further, previous work had shown that allowing the optimization algorithm to update the weights in the embedding layer improves the results of several

NLP tasks [9, 10]. Therefore, we allowed the embedding weights to be adjusted during training.

4.4 TRAINING PROCEDURE

All networks were trained with SGD. Merity et al. [40] pointed out that "for the specific task of neural language modeling, traditionally SGD without momentum has been found to outperform other algorithms such as momentum SGD, Adam, Adagrad and RMSProp by a statistically significant margin".

Following Zaremba et al. [75] we set the initial learning rate to 1 and decreased it after a certain number of epochs using a linear decay rate. We evaluated our models using the average per-word perplexity on a validation set during training and on a test set after training. We terminated the training process when the validation perplexity had stopped improving for five epochs, and kept the model with the best validation perplexity. All models were trained for a maximum number of 50 epochs.

For both datasets, we performed two rounds of random search. In the first round we varied the following parameters: number of hidden neurons, embedding size, decay rate, number of epochs before starting to decay the learning rate. Using existing literature as a basis [39, 40, 75] we chose the following parameter ranges:

- Number of hidden neurons: 200 - 1500 (step size: $200, 300, \ldots$)
- Embedding size: 100 - 500 (step size: $210, 220, \ldots$)
- Decay rate: 0.5 - 0.9 (step size: $0.51, 0.52, \ldots$)
- Number of epochs before starting to decay the learning rate: 5 - 15 (step size: $5, 6, \ldots$)

After finishing the first round of random search, we chose the model with the best validation perplexity for further testing. In the second round of random search the following parameters were varied: embedding dropout probability, LSTM input, recurrent and output dropout probabilities, gradient clipping norm. Once again we used existing literature as a basis for choosing the parameter ranges:

- Embedding, input, recurrent and output dropout probability: 0.0 - 0.5 (step size: $0.01, 0.02, \ldots$)
- Gradient clipping norm: 5 - 10 (step size: $5, 6, \ldots$)

4.5 SENTENCE COMPLETION

To perform sentence completion we initialized a trained model with a word sequence. The final softmax probabilities of the model were used to predict the next word in the sequence, using either the most probable word or sampling from the most probable words, with sample weights proportionate to the word probabilities. The selected word was appended to the input word sequence and the resulting sequence was used as a subsequent input. The prediction process was repeated until the model generated an "end-of-sentence" marker.

4.6 RESULTS MOTORTALK DATASET

4.6.1 *Random Search*

In round one of random search, ten different models were trained and tested. Details on all architectures can be found in the appendix in section A.2.1. The best performing model had 1100 hidden neurons and an embedding size of 250. It was trained with a learning rate of 1.0 for the first seven epochs. After epoch seven the learning rate was decreased by a decay rate of 0.71 per epoch. The model was trained for a total number of 35 epochs. The final validation perplexity was 242.03. In comparison, the worst model achieved validation perplexity 307.56.

The best performing model was selected and further tested in part two of random search, varying the dropout rates and gradient clipping norm. In part two, seven models were tested, two of them using VD instead of standard dropout. The final best performing model (called *motor-baseline* in the following) did not use VD and had the following configurations and performance:

- Number of hidden neurons: 1100
- Embedding size: 250
- Decay rate: 0.71
- Number of epochs before starting to decay the learning rate: 7
- Embedding dropout rate: 0.13
- LSTM input dropout rate: 0.35
- LSTM recurrent dropout rate: 0.27
- LSTM output dropout rate: 0.24
- Gradient clipping norm: 7
- Final validation perplexity: 224.92
- Test perplexity: 218.91

Figure 4.1 illustrates the development of training and validation perplexity of model *motor-baseline*.

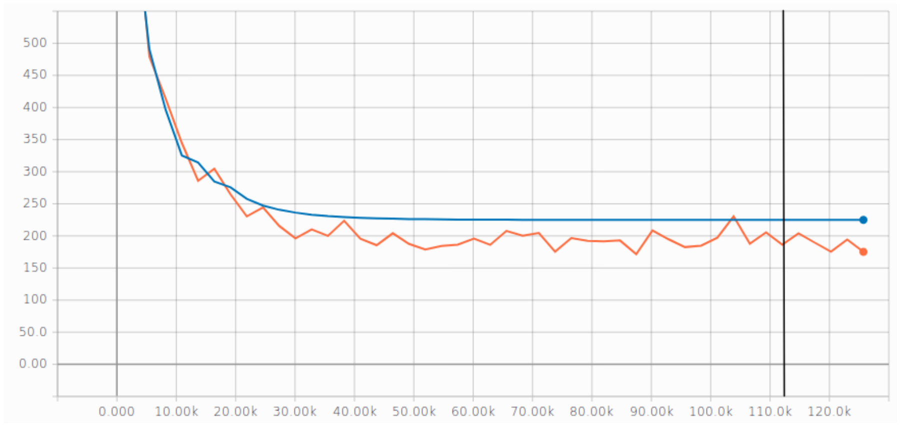


Figure 4.1: Development of training (orange) and validation (blue) perplexity of the best performing model (*motor-baseline*) trained on the motortalk corpus. The x-axis displays the total number of training steps (i. e. updates based on mini-batches). One epoch had 2733 training steps. Training was stopped when the validation accuracy failed to improve for five epochs. The model with the best validation accuracy (see black line) was kept.

4.6.2 *Transfer Learning*

We used the final weights of model *motor-baseline* to initialize a new architecture (*motor-transfer*). The new *motor-transfer* model was trained using the dataset that contained only posts from the BMW forums. As a comparison we trained a third model (*motor-BMW*) whose parameters were initialized by drawing values from a uniform distribution in the interval $[-0.05, 0.05]$. Model *motor-BMW* was also trained solely on data from the BMW forums. It is important to note that this setting is rather unusual for the application of transfer learning. Typically, the dataset from the target domain (which is the BMW dataset in our case) is smaller than the dataset from the source domain. In such transfer learning experiments the researchers typically investigate whether pre-training on the source dataset improves the performance of a model. However, (potentially) observed effects will also be influenced by the fact that the source dataset is bigger than the target dataset. Different to this setting, our source and target datasets had the same size. Therefore, our results were *not* influenced by different dataset sizes.

To ensure that the observed effects were not caused by simply training each model for a larger number of epochs and using a higher learning rate at the beginning of the second training phase, we used the final weights of the *motor-baseline* and *motor-BMW* models to create two new models (*motor-baseline+* and *motor-BMW+*) which were further trained on the mixed- and BMW dataset, respectively. A diagram of the different models and their relationships can be found in figure 4.2. A comparison between the validation and test perplexities

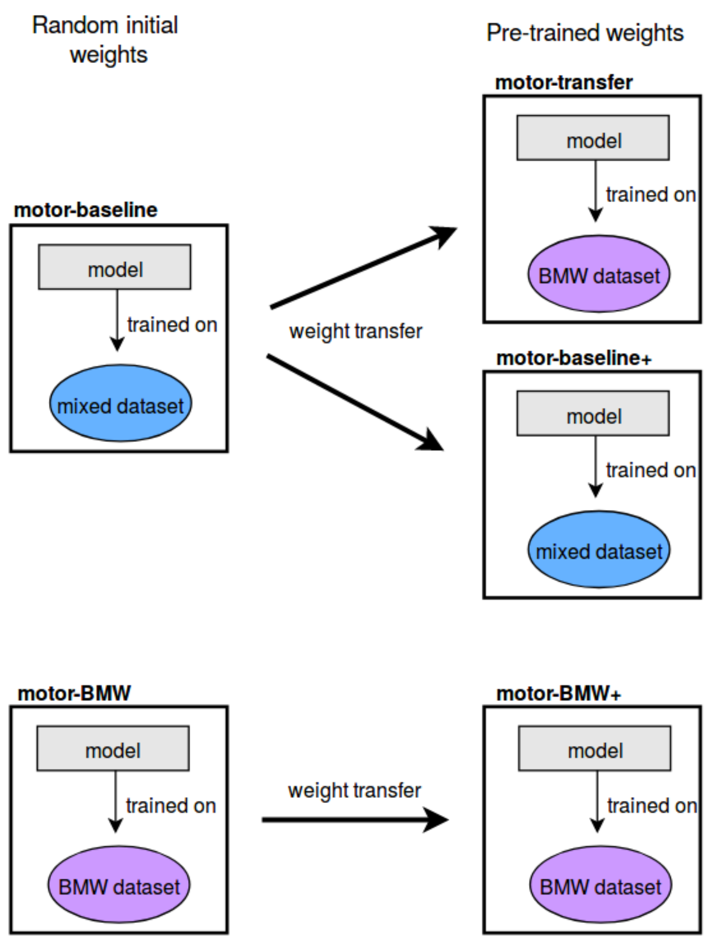


Figure 4.2: Overview of the different models trained on the motortalk dataset

of all models is given in table 4.2. As evident in the table, the second training phase improved the results. In case of the *motor-baseline* model the perplexity was improved by about 6 points from 218.91 to 212.30. In case of *motor-BMW* model the improvement was smaller with just one point in perplexity. The lowest perplexity was achieved by the *motor-transfer* model (210.40).

Model	# Epochs	Train	Valid	Test
motor-baseline	41	204.46	224.92	218.91
motor-baseline+	38	162.29	218.40	212.30
motor-BMW	44	177.81	222.70	253.91
motor-BMW+	44	130.99	217.83	252.85
motor-transfer	45	146.03	183.48	210.40

Table 4.2: Comparison of training, validation and test perplexity of the *motor-baseline* model, *transfer* model and *BMW* model

4.6.3 *Sentence Completion*

To evaluate our models extrinsically (see section 2.5.1) we performed sentence completion. We initialized each model with the primer word "möglicherweise" (English: "possibly") and repeated the prediction process (see section 4.5) until three sentences had been produced. At each time step either the most probable word was appended to the input sequence, or one of the three most probable words was sampled with sample weights proportionate to the word probabilities. Some examples of predicted sentences can be found in table A.5. A detailed overview with predictions of all models can be found in the appendix in section A.2.3.

Model	**Predicted Sentence**
motor-baseline	Möglicherweise nicht. Ich habe mir vor kurzem einen *number* gekauft und habe mir die *number* Zoll Felgen gekauft. Ich habe mir die *number* Felgen gekauft und die sind auch nicht zugelassen.
motor-transfer	Möglicherweise auch nicht. Ich habe das Gefühl dass die *number* im *number* und *number* die gleichen Probleme haben. Die sind auch nicht mehr so gut.

Table 4.3: Comparison of motortalk model predictions given the input sequence "Möglicherweise" (English: "possibly"). For the example of the *motor-baseline* model, at each time step, the most probable word was appended to the input sequence. For the *motor-transfer* example one of the three most probable words was sampled with sample weights proportionate to the word probabilities and appended to the input sequence. Generated end-of-sentence markers were replaced with full stops and capitalization was added manually. The string "number" represents any string that contains a number, like, for example "BMW3".

4.7 RESULTS RADIOLOGICAL DATASET

4.7.1 *Word Embedding Analysis*

Similar to Cornegruta et al. [10] we performed a qualitative assessment of the word embeddings obtained by training a CBOW model on the radiological dataset (embedding size 200). A visualization of the embeddings can be found in figure 4.3. The embeddings were visualized by means of Principal Component Analysis (PCA), using the top three principal components to reduce the dimensionality of the dataset to three dimensions. Furthermore, the data was normalized by shifting each point by the centroid and making it unit norm.

As described by Mikolov et al. [46] learnt word embeddings encode many linguistic regularities and patterns that can be represented as

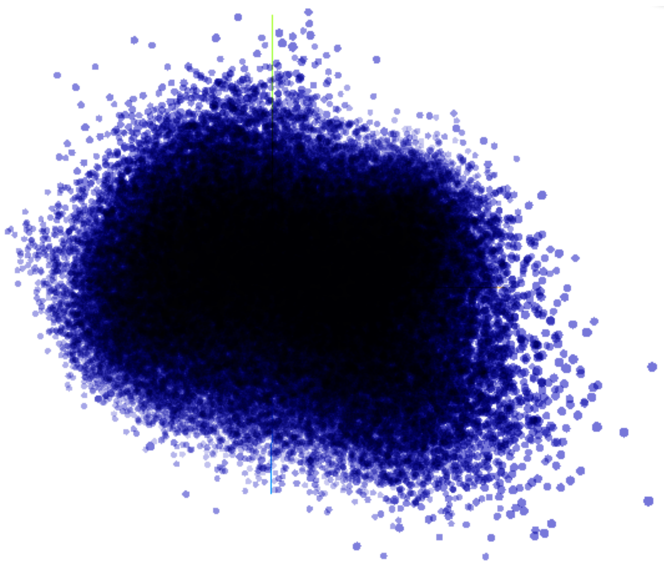


Figure 4.3: Visualization of the embedding space learnt by a CBOW model. The dimensionality of the dataset was reduced with PCA, using the top three principal components. The data was normalized by shifting each point by the centroid and making it unit norm.

linear translations. For example, when summing the vectors of the words "heart" and "enlarged" one of the most similar vectors is "cardiomegaly", which is the medical term for the condition of having an enlarged heart. Similarity in this context is measured by the cosine distance between word vectors. Furthermore, computing the nearest neighbour words reveals the semantic similarity between neighbouring word vectors. For example, the five nearest neighbours of "Blutung" (english: bleeding) are: "Einblutung", "Nachblutung", "Blutungen", "Blutungskomponente", "Einnachblutung".

Also medical relationships are encoded in the embeddings. This becomes evident when comparing the nearest neighbours of the words "Demenz" (English: dementia) and "Alzheimer" (English: Alzheimer's) (see table 4.4). While "Alzheimer" refers to a concrete disease, "Demenz" refers to a group of symptoms. Consequently, the nearest neighbours of "Alzheimer" are diseases, whereas the nearest neighbours of "Demenz" are symptoms or diseases.

The nearest neighbours in the embedding space are further illustrated in figures 4.4 and 4.5.

4.7.2 *Random Search*

Similar to the experiments on the motortalk dataset we trained and tested several different models in round one of random search. Details on all architectures can be found in the appendix in section A.3.2. The best performing model had 1000 hidden neurons and an embed-

demenz	**alzheimer**
depression	pick
psychose	parkinson
schizophrenie	behcet
pnp (polyneuropathie)	addison
epilepsie	menire
polyneuropathie	wegner
ataxie	boeck

Table 4.4: The seven nearest neighbours of the words "Demenz" (dementia) and "Alzheimer" (Alzheimer's) in the embedding space learned by a CBOW model.

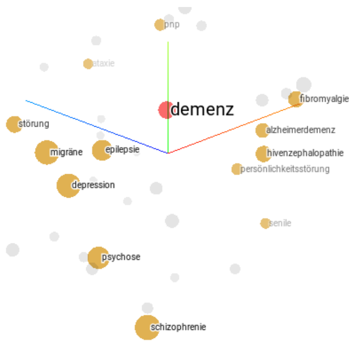


Figure 4.4: Nearest neighbours of the word "Demenz"

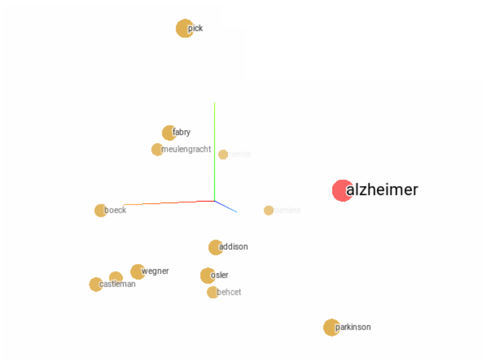


Figure 4.5: Nearest neighbours of the word "Alzheimer"

ding size of 310. It was trained with a learning rate of 1.0 for the first 10 epochs. After epoch 10 the learning rate was decreased by a decay rate of 0.59. The model was trained for a total number of 29 epochs. The final validation perplexity was 6.21.

As before, the best performing model was selected and further tested in part two of random search. In this second part, seven models were tested. The final best model (called *radiology-baseline*) did not use VD and had the following configurations and performance:

- Number of hidden neurons: 1000
- Embedding size: 310
- Decay rate: 0.59
- Number of epochs before starting to decay the learning rate: 10
- Embedding dropout rate: 0.06
- LSTM input dropout rate: 0.33
- LSTM recurrent dropout rate: 0.08

- LSTM output dropout rate: 0.23
- Gradient clipping norm: 9
- Final validation perplexity: 6.2
- Test perplexity: 6.84

Figure 4.6 illustrates the development of training and validation perplexity of model *radiology-baseline*.

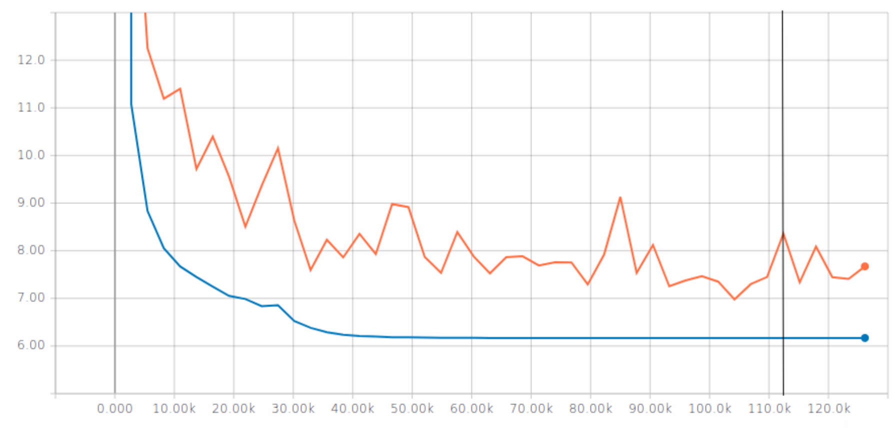


Figure 4.6: Development of training (orange) and validation (blue) perplexity of the best performing model (*radiology-baseline*) trained on the radiological corpus. The x-axis displays the total number of training steps (i. e. updates based on mini-batches). One epoch had 2733 training steps. Training was stopped when the validation accuracy failed to improve for five epochs. The model with the best validation accuracy (see black line) was kept.

4.7.3 *Transfer Learning*

Again, we used the final weights of model *radiology-baseline* to initialize a new architecture (*radiology-transfer-oncology*). The new model was trained solely on reports of patients transferred from the oncological department. As a comparison we trained a third model (*radiology-oncology*) whose parameters were initialized by drawing values from a uniform distribution in the interval $[-0.05, 0.05]$ and that was also trained solely on oncological reports. The same was done for the KTHL dataset, resulting in two more models (*radiology-KTHL* and *radiology-transfer-KTHL*). As mentioned in section 4.6.2, it must be noted that the dataset from the target domain is typically smaller than the dataset from the source domain when applying transfer learning. In our experiments, all datasets had the same size.

Similar to the experiments on the motortalk dataset we trained each model in a second training phase. To do so, we initialized three

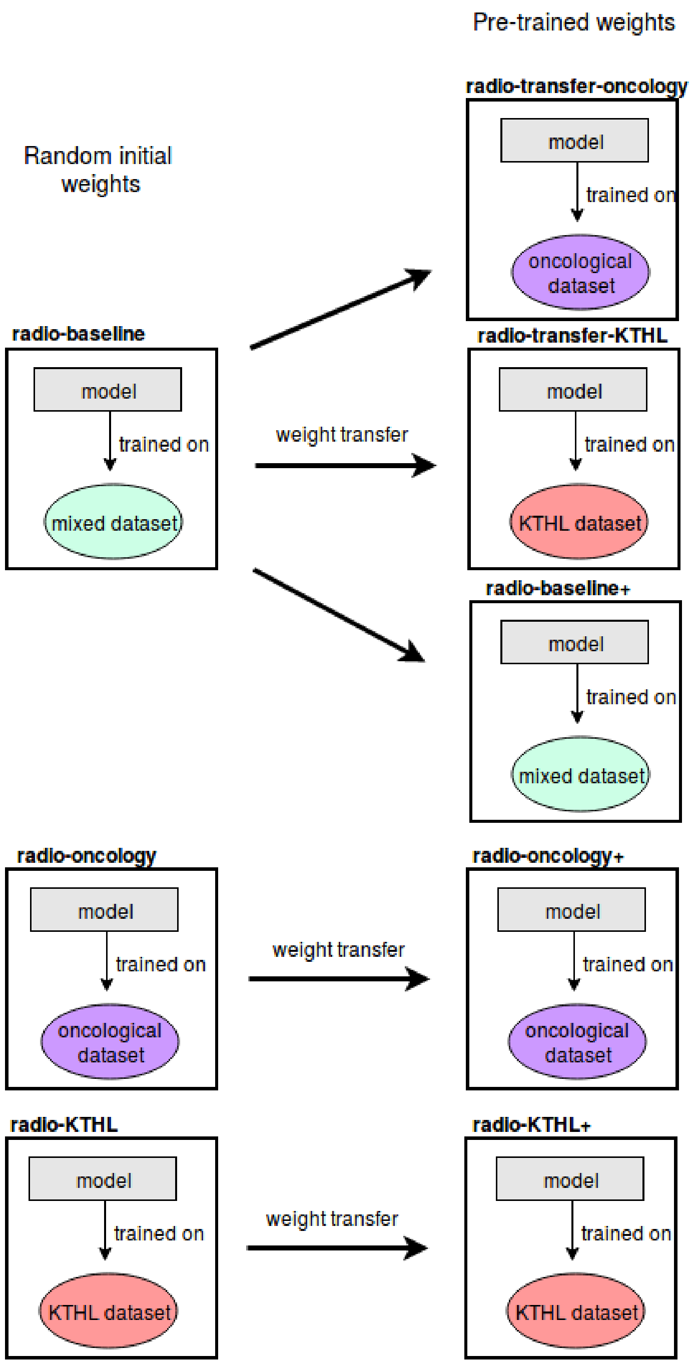


Figure 4.7: Overview of the different models trained on the radiological datasets

models (*radio-baseline+*, *radio-oncology+*, *radio-KTHL+*) using the parameters of the trained baseline, oncology and KTHL model and trained each model using the standard training procedure. These experiments were performed to control for the effects that an increased number of training epochs and an increased learning rate might have. A diagram of the different models and their relationships can be found in figure 4.7. A comparison of all trained models can be found in table 4.5. As shown in the table, the second training phase did not improve the models further. For the models *radio-baseline* and *radio-KTHL* the perplexity remained nearly identical. For the model *radio-oncology+* the perplexity slightly worsened. The two transfer learning models *radio-transfer-oncology* and *radio-transfer-KTHL* performed similarly to the baseline models. The test perplexity of model *radio-transfer-KTHL* was minimally better than the test perplexity of the models *radio-KTHL* and *radio-KTHL+*. The oncology transfer model, however, performed slightly worse than the models *radio-oncology* and *radio-oncology+*.

Model	**# Epochs**	**Train**	**Valid**	**Test**
radio-baseline	41	4.53	6.16	6.72
radio-baseline+	26	4.07	6.17	6.73
radio-oncology	21	10.08	23.20	37.87
radio-oncology+	22	8.77	23.74	38.26
radio-transfer-oncology	21	10.48	23.23	39.96
radio-KTHL	23	4.21	5.79	6.91
radio-KTHL+	38	3.79	5.77	6.95
radio-transfer-KTHL	26	3.79	5.73	6.88

Table 4.5: Comparison of test and validation perplexity of the *radiology-baseline* model, *transfer*, *oncology* and *KTHL* model

4.7.4 *Pre-Trained Embeddings*

To test the effects of using pre-trained word embeddings we trained two more models (*radiology-CBOW* and *radiology-skipgram*) whose embedding matrices were not initialized randomly but with embeddings pre-trained using word2vec. For this purpose, we trained a CBOW and a Skip-gram model using an embedding size of 310 (same embedding size as *radiology-baseline*). The trained input- and output embedding matrices were averaged (as described in section 2.3.1) and the resulting embedding matrices were used to initialize the *radiology-CBOW* and *radiology-skipgram* models. All other initial model parameters were identical to *radiology-baseline*. The models were trained using the same initial learning and decay rate. Also the other training parameters were identical to *radiology-baseline*. Training was stopped when the validation perplexity did not improve for five epochs, and the model with the best validation perplexity was kept. A comparison of the final validation and test perplexities can be found in table 4.6. As evident in the table, all models achieved similar test perplexities. The *radio-skipgram* model performed slightly better (test perplexity: 6.68) than the *radio-cbow* model (test perplexity: 6.85) and *radio-baseline* model (test perplexity: 6.72). However, the differences are minimal.

Model	# Epochs	Train	Valid	Test
radio-baseline	41	4.53	6.16	6.72
radio-CBOW	28	4.90	6.24	6.85
radio-skipgram	26	4.40	6.13	6.68

Table 4.6: Comparison of the training, validation and test perplexity of the *radiology-baseline* model, *transfer*, *word2vec* model and *oncology* model

4.7.5 *Sentence Completion*

Similar to the motortalk-models we evaluated the radiology models by performing sentence completion. We initialized each model with the primer words "es zeigt sich" (English: "the images show") and repeated the prediction process until several sentences had been produced. As before, at each time step either the most probable word was appended to the input sequence, or one of the three most probable words was sampled with sample weights proportionate to the word probabilities. Some examples of predicted sentences can be found in table 4.7. A detailed overview with predictions of all models can be found in the appendix in section A.3.3.

4.8 DISCUSSION

Before discussing specific results a few main differences between the results of the motortalk- and radiology models have to be noted. First, the reported perplexities of models trained on the motortalk dataset are much higher than the reported perplexities of models trained on the radiological dataset. This is caused by the larger number of words in the motortalk vocabulary. Second, for the mixed motortalk dataset, validation perplexities are higher than test perplexities. A possible explanation for this could be that the test set of the mixed motortalk dataset is more similar to the training set than the validation set. For the BMW dataset and the radiological datasets validation perplexities are lower than test perplexities, suggesting the reverse.

For the motortalk dataset our transfer learning approach improved the model's performance. We suspected that the additional training epochs and higher initial learning rate of the second training phase would also yield improvements in perplexity. This turned out to be true. For the mixed motortalk dataset, the second training phase reduced the test perplexity by about 6 points. For the BMW dataset the improvements in test perplexity were only marginal but additional

Model	Predicted Sentence
radiology-transfer-oncology	Es zeigt sich eine deutliche Regredienz der bekannten multiplen Metastasen. Die größte Metastase im rechten Os ilium ist aktuell nicht mehr nachweisbar. Beurteilung *number* im Verlauf unveränderte Darstellung der bekannten multiplen Lebermetastasen. *number* kein Nachweis neu aufgetretener Metastasen. *number* unverändert geringe Menge freier Flüssigkeit im kleinen Becken. *number* unverändert kein Nachweis pathologisch vergrößerter Lymphknoten.
radiology-oncology	Es zeigt sich eine geringe Verdickung der rechten Lunge mit einem Durchmesser vom *number*. Kein Nachweis eines Myelopathiesignals. Kein Nachweis einer Fraktur oder Luxation. Kein Nachweis einer Fraktur oder eines knöchernen Bandausrisses. Regelrechte Darstellung des paravertebralen Weichteilgewebes. Kein Nachweis einer ossären Destruktion des erfassten Skelettsystems. Beurteilung *number* kein Nachweis einer ossären Lymphommanifestation. *number* keine Fraktur.
radiology-KTHL	Es zeigt sich eine homogene Transparenzminderung der rechten Lunge im Sinne eines nach dorsal auslaufenden Ergusses. Kein Pneumothorax. Lagekorrekte Installationen. Röntgen Thorax in einer Ebene auf Station vom *number* im Vergleich zum Vortag zeigt sich eine Zunahme der Lungentransparenz beidseits vor allem rechts basal im Sinne von Abnahme der pulmonalen Infiltrate. Lagekorrekte Installationen kein Pneumothorax. Keine Pleuraergüsse.

Table 4.7: Comparison of model predictions given the input sequence "Es zeigt sich" (English: "the images show"). For the example of the *radiology-transfer-oncology* model, at each time step, the most probable word was appended to the input sequence. For the *radiology-oncology* and *radiology-KTHL* examples one of the three most probable words was sampled with sample weights proportionate to the word probabilities and appended to the input sequence. Generated end-of-sentence markers were replaced with full stops and capitalization was added manually. The string "number" represents any string that contains a number.

training still improved the model's performance. This suggests that the training procedure might benefit from a different learning rate schedule. For example, a cyclic learning rate could be used. In this case, instead of monotonically decreasing the learning rate, it would cyclically vary between different boundary values [59]. Such a cyclic learning rate was shown to provide substantial improvements in performance for several neural network architectures [59].

Although a second training phase improved the motortalk models, the positive effect caused by our transfer learning approach was much larger. The motortalk model using transfer learning (i. e. weights pre-trained on a different corpus) (*motor-transfer*) outperforms the models without transfer learning (*motor-BMW*/*motor-BMW+*) by more than 40 points in perplexity. To test whether this positive effect is caused by fine-tuning on the BMW dataset or simply by training on a more diverse dataset[3] we tested the performance of the model *motor-baseline+* on the BMW test dataset. The resulting test perplexity was 231.83 which is lower than the test perplexity of the *motor-BMW+* model (252.85) but still higher than the test perplexity of the *motor-transfer* model (210.40). This suggests that training on a more diverse dataset first (more diverse regarding vocabulary size and the makes of car talked about) and afterwards fine-tuning the model on the specific dataset yields the largest gains in performance.

The results of the radiological dataset paint a different picture. First, we could not find significant improvements in perplexity when adding a second training phase. A possible explanation for this might be the nature of the radiological datasets. Compared to the motortalk datasets, the vocabulary sizes of the oncological, KTHL, and mixed radiological dataset are much smaller. Consequently, the first training phase might already be sufficient to fit the data. Second, the addition of transfer learning yielded marginal to no improvements. For the oncological dataset our transfer learning approach even increased the perplexity by about one point. We believe that this is caused by the larger vocabulary size of the oncological dataset (35237 words) compared to the mixed radiological dataset (21344 words). In case of the motortalk dataset, the target BMW dataset had a smaller vocabulary size compared to the mixed motortalk dataset. This supports our suggestion that our transfer learning approach improves the performance if the model is trained on a more diverse dataset first. If, however, the dataset of the source task is less diverse than the dataset of the target task, the transferred knowledge in terms of transferred weights is not beneficial. In this context it is important to note that our transfer learning setting is rather unusual. In the traditional setting, in which the target dataset is much smaller than the source dataset, pre-training on a larger but less diverse source dataset might still improve performance.

In case of the KTHL dataset the vocabulary size was smaller (17657 words) compared to the mixed radiological dataset. Still, the improvements caused by transfer learning were only marginal. A possible explanation for this might be that the KTHL dataset is simple enough

3 The vocabulary size of the mixed motortalk dataset (88575 words) was larger than the vocabulary size of the BMW dataset (73704).

that one can already achieve good performance with a single training phase. This is supported by our observation that all KTHL models (*radio-KTHL, radio-KTHL+, radio-transfer-KTHL*) have similar performance.

When testing the effects of initializing the embedding matrix with pre-trained word2vec embeddings we found no improvements in perplexity compared to a random initialization. Although the models with pre-trained embeddings needed fewer epochs to converge, the model with random initial embeddings eventually achieved a similar test perplexity. We believe this to be caused by the fact that word2vec embeddings are trained on the task of predicting context words from target words (or vice versa). In contrast, our models were trained to predict the next word in a sequence of words.

When using variational dropout instead of standard dropout in the experiments on the motortalk dataset, the measured perplexity values were much higher than expected, often exceeding the perplexity values measured for models trained with standard dropout. Similar to the results on the PTB dataset we could not fully explain these effects. Because of the high perplexities observed for the motortalk dataset we did not test variational dropout for the radiological dataset.

Regarding the sentence completion task, we gave all generated radiological reports listed in the appendix (see section A.3.3) to radiologists from Essen University Hospital and asked for their judgement on validity (with regard to content) and quality. All reports from the models *radiology-baseline* and *radiology-baseline+* were identified as valid descriptions of findings related to an X-ray of a patient's thorax. It was further pointed out that an X-ray of the thorax is the most common radiological examination. As mentioned in section 4.2 the KTHL dataset contains only reports for which the radiological examination was given by an X-ray of the patient's thorax while the patient was lying down. Consequently, the reports generated by the models *radiology-KTHL*, *radiology-KTHL+* and *radiology-KTHL-transfer* were reported to be very similar to the baseline models.

The reports of models *radiology-oncology*, *radiology-oncology+* and *radiology-oncology-transfer*, especially those generated by sampling from the most probable words, were reported to be of lower quality compared to the baseline and KTHL models (for an example see comparison *radiology-oncology* vs. *radiology-KTHL* in table 4.7). This could be caused by the larger vocabulary size of the oncological dataset.

The word sequences generated by the motortalk models are of a colloquial, forum-typical language: sentences are short and contain simple words and grammar. Also, they sometimes contain repetitions of words or sequences of words. Still, most sentences are valid.

5 CONCLUSION AND FUTURE WORK

In this thesis, we investigated the abilities of word-based RNNLMs to model radiological reports. Further, to test the applicability of our techniques to other domains, we performed experiments on a corpus of forum posts on motorized vehicles.

In our experiments, we tested the potential benefits of several advanced regularization and initialization techniques, including embedding dropout, variational dropout, transfer learning, weight tying and pre-trained word embeddings. We were able to show that basic RNNLMs already perform well when being trained on the task of predicting the next word in a sequence of words, but that their performance can be improved by tuning hyperparameters and by adding advanced regularization techniques such as embedding dropout or weight tying. Furthermore, our results suggest that the application of transfer learning can yield substantial improvements in perplexity if the datasets are complex enough in terms of vocabulary size and structure of the dataset and if the dataset of the source task is more diverse than the dataset of the target task.

Regarding the desired final application, that is, accelerating the composition of radiological reports, we showed that our models are able to produce convincing radiological reports. This suggests their suitability for being applied in praxis and, possibly, to other medical domains.

In addition to our experiments on RNNLMs, we analyzed medical word embeddings obtained by training a CBOW model on the radiological dataset. In this context we made three key observations. First, we found that the embeddings encode linguistic regularities and patterns which can be represented as linear translations of word vectors. Second, we found that the words (or more precisely the word vectors) closest to some word in the embedding space are semantically similar to that word. Third, we observed that medical relationships are encoded in the embeddings.

Several directions for future work remain to be explored. As already mentioned in the introduction, radiological reports contain many synonyms and abbreviations as well as grammar and spelling mistakes. Correcting the mistakes and normalizing the synonyms, abbreviations and Latin spelling variations would likely improve our models' predictions. The learned medical word embedding vectors could be helpful for this task. Another benefit concerns the observed relations between embedding vectors. The quality of a word vector depends both on the number of times the corresponding word appears in a corpus, and the variety of contexts the word appears in. When normalizing synonyms and correcting mistakes, the vocabulary size would decrease. As a consequence, the remaining words would occur more often and in a wider variety of contexts, decreasing the number of possible word transitions a model has to learn and improving both the quality of the word vectors and the observed relations between word vectors.

Further improvements could be achieved by combining the best performing models into an ensemble instead of using only a single model.

In addition to an advanced preprocessing procedure and ensemble learning, we are planning to investigate other ways to incorporate context into the networks. For example, a network could be given information about the department from which a patient was transferred or what kind of radiological examination was performed. This would allow the model to adapt its suggestions accordingly. Also, we plan to evaluate our system by employing it in the radiological department of Essen University Hospital. For this purpose, we will train a model on the entire corpus of radiological reports, containing more than 220 million words. We expect that this large amount of training data will further improve the results. As outlined in the introduction, we believe that the successful application of our system would yield major benefits. First and foremost, large amounts of time and effort would be saved. In addition, the language of the reports would be less variable, containing fewer synonyms and spelling mistakes. This would help to standardize reports and make it easier for non-physicians to understand them. Finally, the system could be applied to other medical disciplines, potentially accelerating any procedure that involves the preparation of written reports.

A
APPENDIX

A.1 RESULTS PENN TREEBANK DATASET

As mentioned in section 4.1, we performed initial experiments on the PTB data set. We trained several small (200 units) and medium (650 units) models with different architectural features and different dropout rates. Table A.1 lists the word-level perplexities of the small models on the validation and test set, table A.2 lists the medium models. Details about hyperparameters and the general model architecture can be found in section 4.1.

Embedding Drop	**Input Drop**	**Recurrent Drop**	**Output Drop**	**Tied weights**	**Variational dropout**	Valid	Test
						122.81	120.66
	0.3		0.3			99.33	96.14
	0.1		0.1			108.05	103.76
	0.3		0.3	✓		99.15	95.33
	0.1		0.1	✓		100.12	96.02
0.3	0.3		0.3			106.03	102.95
0.1	0.1		0.1	✓		95.7	91.79
	0.3	0.3	0.3		✓	104.01	99.40
	0.2	0.2	0.2		✓	99.09	94.83
	0.1	0.1	0.1		✓	101.24	97.98
	0.2	0.2	0.2	✓	✓	101.6	97.03
	0.1	0.1	0.1	✓	✓	98.30	94.28
0.1	0.1	0.1	0.1	✓	✓	96.98	92.98

Table A.1: Comparison of the word-level perplexities on the validation and test set for small models (200 units) with different architectural features tested on the PTB dataset

Embedding Drop	Input Drop	Recurrent Drop	Output Drop	Tied weights	Variational dropout	Valid	Test
	0.5		0.5			86.11	82.88
	0.3		0.3			107.37	104.14
	0.5		0.5	✓		82.17	79.04
0.2	0.5		0.5	✓		85.32	82.24
0.1	0.3		0.3	✓		83.15	80.41
0.2	0.5		0.5			85.69	82.46
	0.3	0.3	0.3		✓	86.97	83.81
	0.5	0.3	0.5		✓	686.95	639.65
0.1	0.3	0.3	0.3		✓	686.76	639.57
	0.3	0.3	0.3	✓	✓	82.67	79.73
	0.5	0.3	0.5	✓	✓	95.29	91.51
0.1	0.3	0.3	0.3	✓	✓	82.0	78.64
	0.5		0.5	✓	✓	99.13	95.72
0.1	0.5	0.3	0.3	✓	✓	84.29	81.24
0.1	0.4	0.3	0.4	✓	✓	87.17	83.96

Table A.2: Comparison of the word-level perplexities on the validation and test set for medium models (650 units) with different architectural features tested on the PTB dataset

A.2 MOTORTALK DATASET

A.2.1 *Random Search Part One*

In part one of random search the following parameters were varied: number of hidden neurons, embedding size, decay rate, number of epochs after which to start decaying. The allowed ranges of the parameters can be found in section 4.4. All models were trained with tied weights but without embedding or variational dropout. The input and output dropout probabilities of the LSTM layers were fixed to 0.3 for all models tested in this part of random search. The maximal gradient clipping norm was fixed to 5.

# Hidden Neurons	**Embedding Size**	**Decay Rate**	**Decay Epoch**	**# Epochs**	Train	Valid
400	310	0.63	6	27	163.14	244.39
400	120	0.68	7	32	241.12	285.17
300	330	0.72	13	50	124.94	253.08
1100	330	0.65	12	47	115.34	243.38
1100	250	0.71	7	35	146.48	242.03
300	100	0.57	6	27	291.38	307.56
500	290	0.79	9	15	144.26	249.72
200	260	0.69	11	38	163.5	253.70
700	180	0.75	7	41	173.52	258.70
1400	300	0.86	9	12	145.54	248.83

Table A.3: Overview of models tested on the motortalk corpus in part one of random search. The following parameters were varied: number of hidden neurons, embedding size, decay rate, number of epochs after which to start decaying the learning rate. The table also lists the total number of epochs the models were trained before the validation perplexity stopped improving. The best model is highlighted in blue.

A.2.2 *Random Search Part Two*

In part two of random search the following parameters were varied: embedding dropout probability, LSTM input and recurrent and output dropout probabilities, gradient clipping norm and whether or not variational dropout was used for the LSTM units. The allowed ranges of the parameters can be found in section 4.4. The model architecture was chosen according to the best model from part one of random search. Consequently, the number of neurons was set to 1100 and the embedding size to 250. The learning rate was decayed after 7 epochs with a decay rate of 0.71.

Embedding Drop	**Input Drop**	**Recurrent Drop**	**Output Drop**	**VD**	**Clip Norm**	**Number Epochs**	Train	Valid	Test
0.19	0.18	0.21	0.2		9	37	365.35	307.11	300.11
0.25	0.45	0.23	0.39	✓	9	49	701.92	580.18	568.64
0.13	0.12	0.39	0.4	✓	6	49	734.79	648.67	636.41
0.16	0.38	0.0	0.44		10	45	354.57	294.34	286.91
0.14	0.32	0.43	0.38		9	24	226.81	235.07	228.77
0.13	0.35	0.27	0.24		7	41	204.46	224.92	218.91
0.2	0.33	0.19	0.31		5	48	271.25	239.03	232.97

Table A.4: Overview of models tested on the motortalk corpus in part two of random search. The following parameters were varied: embedding dropout probability, LSTM input, recurrent and output dropout probabilities, gradient clipping norm and whether or not variational dropout was used for the LSTM units. The table also lists the total number of epochs the models were trained before the validation perplexity stopped improving. The best model is highlighted in blue.

A.2.3 *Sentence Completion*

To evaluate our models extrinsically (see section 2.5.1) we performed sentence completion. We initialized the model with the primer word "möglicherweise" (English: "possibly") and repeated the prediction process (see section 4.5) until three sentences had been produced. At each time step either the most probable word was appended to the input sequence, or one of the three most probable words was sampled with sample weights proportionate to the word probabilities.

A.3 RADIOLOGICAL DATASET

A.3.1 *Random Search Part One*

In part one of random search the following parameters were varied: number of hidden neurons, embedding size, decay rate, number of epochs after which to start decaying. The allowed ranges of the parameters can be found in section 4.4. All models were trained with tied weights but without embedding or variational dropout. The input and output dropout probabilities of the LSTM layers were fixed to 0.3 for all models tested in this part of random search. The maximal gradient clipping norm was fixed to 5.

Model	**Predicted Sentence**
motor-baseline (best word)	Möglicherweise nicht. Ich habe mir vor kurzem einen *number* gekauft und habe mir die *number* Zoll Felgen gekauft. Ich habe mir die *number* Felgen gekauft und die sind auch nicht zugelassen.
motor-baseline (sampled word)	Möglicherweise nicht so schlimm. Das ist doch ein *rare*. Ich hab das Auto auch noch nicht gesehen.
motor-baseline+ (best word)	Möglicherweise ist das Auto nicht mehr so teuer wie ein *number* und das ist nicht so schlimm. Ich habe mir vor kurzem einen *number* mit *number* gekauft. Ich habe mir vor kurzem einen *number* CDI gekauft.
motor-baseline+ (sampled word)	Möglicherweise ist das Auto auch ein *number* und ich würde mich auch sehr ärgern dass ich das Auto nicht mit *number* fahren will. Das Auto ist auch nicht so schlecht und ist auch sehr gut. Ich bin mit dem *number* auch zufrieden mit der *number* Maschine.
motor-BMW (best word)	Möglicherweise auch nicht. Ich habe das Gefühl dass ich den *number* mit *number* ps mit *number* ps fahre. Ich habe den *number* mit *number* ps und *number* ps.
motor-BMW (sampled word)	Möglicherweise nicht. Das ist doch auch so ein *rare*. Das ist doch auch nicht der Einzige der das ganze mal mit der Zeit in der Garage fahren kann.
motor-BMW+ (best word)	Möglicherweise ist das System nicht so gut wie ein *number* das ist ein echter Witz. Ich habe das Auto auch schon seit *number* Jahren gekauft und bin begeistert. Ich habe mir einen *number* gekauft und bin begeistert.
motor-BMW+ (sampled word)	Möglicherweise nicht. Ich habe das Auto seit *number* Jahren mit *number* gekauft und habe mich dann mit dem Verkäufer gefragt. Der Verkäufer hat mir gesagt ich habe ihn nicht mehr gefahren. Der Wagen ist jetzt auch noch nicht mehr so schlecht.
motor-transfer (best word)	Möglicherweise nicht. Ich habe mir jetzt einen *number* gekauft. Ich habe mir einen *number* gekauft und bin zufrieden.
motor-transfer (sampled word)	Möglicherweise auch nicht. Ich habe das Gefühl dass die *number* im *number* und *number* die gleichen Probleme haben. Die sind auch nicht mehr so gut.

Table A.5: Comparison of model predictions given the input sequence "möglicherweise" (English: "possibly"). At each time step either the most probable word was appended to the input sequence, or one of the three most probable words was sampled with sample weights proportionate to the word probabilities. Generated end-of-sentence markers were replaced with full stops and capitalization was added manually. The string "number" represents any string that contains a number, like, for example "BMW3".

# Hidden Neurons	**Embedding Size**	**Decay Rate**	**Decay Epoch**	**# Epochs**	Train	Valid
1000	310	0.59	10	29	4.25	6.21
700	250	0.58	7	25	4.7	6.40
1100	250	0.71	7	38	4.46	6.27
800	200	0.81	12	50	4.37	6.33
600	390	0.69	7	26	4.12	6.26
1300	180	0.67	9	33	4.61	6.44
1400	180	0.86	14	35	3.96	6.49
230	200	0.56	10	34	5.58	6.9

Table A.6: Overview of models tested on the radiological corpus in part one of random search. The following parameters were varied: number of hidden neurons, embedding size, decay rate, number of epochs after which to start decaying the learning rate. The table also lists the total number of epochs the models were trained before the validation perplexity stopped improving. The best model is highlighted in blue.

A.3.2 *Random Search Part Two*

In part two of random search the following parameters were varied: embedding dropout probability, LSTM input and recurrent and output dropout probabilities, gradient clipping norm. The allowed ranges of the parameters can be found in section 4.4. The model architecture was chosen according to the best model from part one of random search. Consequently, the number of neurons was set to 1000 and the embedding size to 310. The learning rate was decayed after 10 epochs with a decay rate of 0.59.

Embedding Drop	**Input Drop**	**Recurrent Drop**	**Output Drop**	**Clip Norm**	**Number Epochs**	Train	Valid	Test
0.06	0.33	0.08	0.23	5	41	4.53	6.16	6.72
0.05	0.29	0.42	0.46	7	34	10.19	8.55	8.9
0.26	0.17	0.35	0.46	10	28	8.86	7.22	7.68
0.28	0.35	0.08	0.38	9	19	8.17	7.05	7.55
0.44	0.09	0.12	0.16	5	27	8.18	6.53	7.15
0.42	0.2	0.06	0.27	6	31	8.9	6.78	7.41
0.13	0.32	0.2	0.35	5	29	5.82	6.35	6.89

Table A.7: Overview of models tested on the radiological corpus in part two of random search. The following parameters were varied: embedding dropout probability, LSTM input, recurrent and output dropout probabilities, gradient clipping norm. The table also lists the total number of epochs the models were trained before the validation perplexity stopped improving. The best model is highlighted in blue.

A.3.3 *Sentence Completion*

To evaluate our models extrinsically (see section 2.5.1) we performed sentence completion. We initialized the model with the primer words "es zeigt sich" (English: "the images show") and repeated the prediction process (see section 4.5) until three sentences had been produced. At each time step either the most probable word was appended to the input sequence, or one of the three most probable words was sampled with sample weights proportionate to the word probabilities.

Model	**Predicted Sentence**
radiology-baseline (best word)	Es zeigt sich eine geringe Zunahme der pulmonalen Gefäßfüllung. Keine akute kardiale Dekompensation. Keine entzündlichen Infiltrate. Verlaufskontrolle Röntgen des Thorax in einer Ebene auf der Station vom *number* im Vergleich zum Vortag zeigt sich eine Befundverschlechterung im Sinne von Entwicklung von diffusen homogenen Transparenzminderung beidseits basal rechts links mit Nachweis von positivem Aerobronchogramm im Sinne von Infiltraten. Keine Pleuraergüsse.
radiology-baseline (sample best words)	Es zeigt sich eine homogene Trübung beider Lungen mit Betonung der Herz und Zwerchfellgrenzen. Keine Ergüsse. Kein Pneumothorax nachweisbar. Zn. acb. Verlaufskontrolle Röntgen des Thorax in einer Ebene auf der Station vom *number* im Vergleich zum Vortag zeigt sich eine Befundverschlechterung im Sinne einer Entwicklung von homogenen Transparenzminderung rechts basal mit Nachweis von positivem Aerobronchogramm.
radiology-baseline+ (best word)	Es zeigt sich eine geringe Zunahme der pulmonalen Gefäßfüllung. Keine akute kardiale Dekompensation. Keine entzündlichen Infiltrate. Verlaufskontrolle Röntgen des Thorax in einer Ebene auf der Station vom *number* im Vergleich zum Vortag zeigt sich eine Befundverschlechterung im Sinne von Entwicklung von diffusen homogenen Transparenzminderung beidseits basal rechts links mit Nachweis von positivem Aerobronchogramm im Sinne von Infiltraten. Keine Pleuraergüsse.
radiology-baseline+ (sample best words)	Es zeigt sich eine leichte Befundverschlechterung mit zunehmender interstitieller Zeichnungsvermehrung im Sinne einer interstitiellen Wassereinlagerung. Kein Pneumothorax. Lagekorrekte Installationen. Verlaufskontrolle Röntgen des Thorax in einer Ebene auf der Station vom *number* im Vergleich zum Vortag zeigt sich eine Befundverschlechterung im Sinne von Zunahme der pulmonalen Infiltrate beidseits. Keine Stauungszeichen. Kein Pleuraerguss. Kein Pneumothorax. Lagekorrekte Installation.

Table A.8: Comparison of *radiology-baseline* and *radiology-baseline+* predictions given the input sequence "Es zeigt sich" (English: "the images show"). At each time step either the most probable word was appended to the input sequence, or one of the three most probable words was sampled with sample weights proportionate to the word probabilities. Generated end-of-sentence markers were replaced with full stops and capitalization was added manually. The string "number" represents any string that contains a number.

Model	Predicted Sentence
radiology-oncology (best word)	Es zeigt sich eine deutliche Signalanhebung in der Stir. Die Hinterkante ist nicht relevant eingeengt. Kein Nachweis einer intraspinalen Raumforderung. Beurteilung *number* kein Nachweis einer intraspinalen Tumormanifestation. *number* kein Nachweis einer intraspinalen Tumormanifestation.
radiology-oncology (sample best words)	Es zeigt sich eine geringe Verdickung der rechten Lunge mit einem Durchmesser vom *number*. Kein Nachweis eines Myelopathiesignals. Kein Nachweis einer Fraktur oder Luxation. Kein Nachweis einer Fraktur oder eines knöchernen Bandausrisses. Regelrechte Darstellung des paravertebralen Weichteilgewebes. Kein Nachweis einer ossären Destruktion des erfassten Skelettsystems. Beurteilung *number* kein Nachweis einer ossären Lymphommanifestation. *number* keine Fraktur.
radiology-oncology+ (best word)	Es zeigt sich eine geringe Verdickung der Knochenstruktur. Die übrigen abgebildeten Wirbelkörper stellen sich unauffällig dar. Beurteilung *number* kein Nachweis einer intraspinalen Tumormanifestation. *number* keine Spinalkanalstenose.
radiology-oncology+ (sample best words)	Es zeigt sich eine geringe Verdickung der Knochenstruktur des linken m Psoas. Das rechte ISG stellt sich unauffällig dar. Kein Nachweis einer Fraktur oder einer akuten Fraktur. Kein Anhalt für eine frische knöcherne Verletzung. Beurteilung *number* keine Fraktur. *number* kein Anhalt für eine frische knöcherne Verletzung *number* kein Nachweis einer ossären Metastasierung *number* keine ossären Metastasen.

Table A.9: Comparison of *radiology-oncology* and *radiology-oncology+* predictions given the input sequence "Es zeigt sich" (English: "the images show"). At each time step either the most probable word was appended to the input sequence, or one of the three most probable words was sampled with sample weights proportionate to the word probabilities. Generated end-of-sentence markers were replaced with full stops and capitalization was added manually. The string "number" represents any string that contains a number.

Model	Predicted Sentence
radiology-transfer-oncology (best word)	Es zeigt sich eine deutliche Regredienz der bekannten multiplen Metastasen. Die größte Metastase im rechten Os ilium ist aktuell nicht mehr nachweisbar. Beurteilung *number* im Verlauf unveränderte Darstellung der bekannten multiplen Lebermetastasen. *number* kein Nachweis neu aufgetretener Metastasen. *number* unverändert geringe Menge freier Flüssigkeit im kleinen Becken. *number* unverändert kein Nachweis pathologisch vergrößerter Lymphknoten.
radiology-transfer-oncology (sample best words)	Es zeigt sich es zeigt sich eine geringe Zunahme der interstitiellen Verdichtungen im Sinne eines Lungenödems mit geringen Begleitergüssen. Keine Infiltrate abgrenzbar. Kein Pneumothorax. Keine Atelektasen. Herz und Mediastinum lagebedingt verbreitert. aml Fieber in Neutropenie Infiltrate. Ergüsse. Röntgen Thorax liegend in *number* Ebene vom *number* Befund und Beurteilung im Vergleich zur Voruntersuchung vom *number* neu abgrenzbare flächige Transparenzminderung rechts basal.

Table A.10: Comparison of *radiology-transfer-oncology* predictions given the input sequence "Es zeigt sich" (English: "the images show"). At each time step either the most probable word was appended to the input sequence, or one of the three most probable words was sampled with sample weights proportionate to the word probabilities. Generated end-of-sentence markers were replaced with full stops and capitalization was added manually. The string "number" represents any string that contains a number.

Model	**Predicted Sentence**
radiology-KTHL (best word)	Es zeigt sich eine homogene Transparenzminderung der rechten Lunge im Sinne eines nach dorsal auslaufenden Ergusses. Kein Pneumothorax. Lagekorrekte Installationen. Röntgen Thorax in einer Ebene auf Station vom *number* im Vergleich zum Vortag zeigt sich eine Zunahme der Lungentransparenz beidseits vor allem rechts basal im Sinne von Abnahme der pulmonalen Infiltrate. Lagekorrekte Installationen kein Pneumothorax. Keine Pleuraergüsse.
radiology-KTHL (sample best words)	Es zeigt sich eine deutliche Transparenzminderung rechts basal im Sinne einer Minderbelüftung. Weiterhin keine entzündlichen Infiltrate. Kein Pneumothorax. Herz und Mediastinalbreite unverändert. Röntgen Thorax in einer Ebene auf Station vom *number* im Vergleich zur Voruntersuchung von heute *number* zn. Entfernung der Magensonde sowie der Schleuse von rechts. Kein Nachweis eines Pneumothorax. Unverändert einliegende Installationen.
radiology-KTHL+ (best word)	Es zeigt sich es zeigt sich eine homogene Transparenzminderung rechts basal im sinne eines Pleuraergusses. Lagekorrekte Installationen. Kein Pneumothorax. Keine kardiale Dekompensation. Keine entzündlichen Infiltrate. Verlaufskontrolle Röntgen Thorax in einer Ebene auf Station vom *number* im Vergleich zum Vortag zeigt sich eine Zunahme der Lungentransparenz beidseits vor allem rechts basal im Sinne von Entwicklung von interstitiellen Infiltraten.
radiology-KTHL+ (sample best words)	Es zeigt sich eine homogene Transparenzminderung der gesamten rechten Lunge mit Nachweis von positivem Aerobronchogramm im Sinne einer Minderbelüftung. Kein Pneumothorax. Keine Pleuraergüsse. Keine kardiale Dekompensation. Verlaufskontrolle Röntgen Thorax in einer Ebene auf Station vom *number* im Vergleich zum Vortag zeigt sich bei unterschiedlicher rare keine Befundänderung. Unverändert lagekorrekte Installationen. Kein Pneumothorax. Keine kardiale Dekompensation.

Table A.11: Comparison of *radiology-KTHL* and *radiology-KTHL+* predictions given the input sequence "Es zeigt sich" (English: "the images show"). At each time step either the most probable word was appended to the input sequence, or one of the three most probable words was sampled with sample weights proportionate to the word probabilities. Generated end-of-sentence markers were replaced with full stops and capitalization was added manually. The string "number" represents any string that contains a number.

Model	Predicted Sentence
radiology-transfer-KTHL (best word)	Es zeigt sich eine diffuse homogene Transparenzminderung beider Lungen mit Betonung der pulmonalen Gefäßfüllung. Dies alles spricht für eine cardiopulmonale Dekompensation mit beginnender Zunahme der pulmonalen Gefäßfüllung. Kein Pneumothorax. Keine Pleuraergüsse. Keine Atelektase. Keine entzündlichen Infiltrate. Verlaufskontrolle Röntgen des Thorax in einer Ebene auf der Station vom *number* im Vergleich zum Vortag zeigt sich eine Befundverschlechterung.
radiology-transfer-KTHL (sample best words)	Es zeigt sich eine deutliche Zunahme des basalen Pneumothorax mit aktuell regelrechter Lage der *rare*. Unverändert lagekorrekte Installationen. Keine entzündlichen Infiltrate. Verlaufskontrolle Röntgen des Thorax in einer Ebene auf der Station vom *number* im Vergleich zum heutigen *rare* keine Befundänderung. Keine Stauungs oder Entzündungszeichen pulmonal kein Pneumothorax oder Atelektasen. Lagekorrekte Installationen. Postop.

Table A.12: Comparison of *radiology-transfer-KTHL* predictions given the input sequence "Es zeigt sich" (English: "the images show"). At each time step either the most probable word was appended to the input sequence, or one of the three most probable words was sampled with sample weights proportionate to the word probabilities. Generated end-of-sentence markers were replaced with full stops and capitalization was added manually. The string "number" represents any string that contains a number.

Model	Predicted Sentence
radiology-CBOW (best word)	Es zeigt sich eine homogene Transparenzminderung der linken Lunge im Sinne eines progredienten Pleuraergusses. Kein Pneumothorax. Lagekorrekte Installationen. Verlaufskontrolle Röntgen Thorax in einer Ebene auf Station vom *number* im Vergleich zum Vortag zeigt sich eine Zunahme der Lungentransparenz beidseits im Sinne einer Abnahme der cardiopulmonalen Stauung. Keine Pleuraergüsse.
radiology-CBOW (sample best words)	Es zeigt sich ein zn. Intubation. Der ett projeziert sich mit seiner Spitze auf die Carina. Rückzug um *number* cm empfohlen. Einliegende Magensonde. Kein Pneumothorax. Keine Pleuraergüsse. Röntgen Thorax in einer Ebene auf Station vom *number* im Vergleich zur Voraufnahme vom *number* kardiopulmonal Status idem. Herz und Mediastinalschatten lagebedingt verbreitert. Keine pulmonalen Stauungszeichen.
radiology-skipgram (best word)	Es zeigt sich eine geringe Zunahme der pulmonalen Gefäßfüllung. Keine kardiale Dekompensation. Keine entzündlichen Infiltrate. Kein Pneumothorax. Röntgen des Thorax in einer Ebene auf der Station vom *number* im Vergleich zum Vortag zeigt sich eine Zunahme der Lungentransparenz beidseits im Sinne von Abnahme der pulmonalen Gefäßfüllung. Keine akute kardiale Dekompensation. Keine Pleuraergüsse. Lagekorrekte Installationen kein Pneumothorax.
radiology-skipgram (sample best words)	Es zeigt sich eine geringe Transparenzminderung der linken Lunge vereinbar mit einer Minderbelüftung. Kein Pneumothorax. Mittelständiges nicht verbreitertes oberes Mediastinum. Normal großes Herz. Zn. Extubation und Entfernung der Magensonde. Zn. acvb Verlauf Röntgen Thorax in einer Ebene auf Station vom *number* im Vergleich zur Voruntersuchung vom Vortag unveränderter kardiopulmonaler Befund. Kein Pneumothorax.

Table A.13: Comparison of *radiology-CBOW* and *radiology-skipgram* predictions given the input sequence "Es zeigt sich" (English: "the images show"). At each time step either the most probable word was appended to the input sequence, or one of the three most probable words was sampled with sample weights proportionate to the word probabilities. Generated end-of-sentence markers were replaced with full stops and capitalization was added manually. The string "number" represents any string that contains a number.

BIBLIOGRAPHY

[1] Yoshua Bengio, Paolo Frasconi, and Patrice Simard. "The problem of learning long-term dependencies in recurrent networks." In: *Neural Networks, 1993., IEEE International Conference on*. IEEE. 1993, pp. 1183–1188.

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

[3] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." In: *Journal of machine Learning research* 3.Jan (2003), pp. 993–1022.

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. "Enriching word vectors with subword information." In: *arXiv preprint arXiv:1607.04606* (2016).

[5] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. "One billion word benchmark for measuring progress in statistical language modeling." In: *arXiv preprint arXiv:1312.3005* (2013).

[6] Stanley F. Chen and Joshua Goodman. "An empirical study of smoothing techniques for language modeling." In: *Proceedings of the 34th annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics. 1996, pp. 310–318.

[7] Billy Chiu, Gamal Crichton, Anna Korhonen, and Sampo Pyysalo. "How to Train Good Word Embeddings for Biomedical NLP." In: *Proceedings of the 15th Workshop on Biomedical Natural Language Processing* (2016), pp. 166–174.

[8] Ronan Collobert and Jason Weston. "A unified architecture for natural language processing: Deep neural networks with multitask learning." In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 160–167.

[9] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. "Natural language processing (almost) from scratch." In: *Journal of Machine Learning Research* 12.Aug (2011), pp. 2493–2537.

[10] Savelie Cornegruta, Robert Bakewell, Samuel Withey, and Giovanni Montana. "Modelling radiological language with bidirectional long short-term memory networks." In: *arXiv preprint arXiv:1609.08409* (2016).

[11] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. "Indexing by latent semantic analysis." In: *Journal of the American society for information science* 41.6 (1990), p. 391.

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

[13] Jeffrey L. Elman. "Finding structure in time." In: *Cognitive science* 14 (1990), pp. 179–211.

[14] Yarin Gal and Zoubin Ghahramani. "A theoretically grounded application of dropout in recurrent neural networks." In: *Advances in neural information processing systems*. 2016, pp. 1019–1027.

[15] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, 2017.

[16] Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. "Learning precise timing with LSTM recurrent networks." In: *Journal of machine learning research* 3.Aug (2002), pp. 115–143.

[17] Jen J. Gong, Tristan Naumann, Peter Szolovits, and John V. Guttag. "Predicting Clinical Outcomes Across Changing Electronic Health Record Systems." In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2017, pp. 1497–1505.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[19] Joshua T. Goodman. "A Bit of Progress in Language Modeling Extended Version." In: *Computer Speech & Language* 15.4 (2001), pp. 403–434.

[20] Alex Graves. "Generating sequences with recurrent neural networks." In: *arXiv preprint arXiv:1308.0850* (2013).

[21] Alex Graves and Navdeep Jaitly. "Towards end-to-end speech recognition with recurrent neural networks." In: *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. 2014, pp. 1764–1772.

[22] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks." In: *IEEE international conference on Acoustics, speech and signal processing (ICASSP)*. IEEE. 2013, pp. 6645–6649.

[23] Alex Graves and Jürgen Schmidhuber. "Offline handwriting recognition with multidimensional recurrent neural networks." In: *Advances in neural information processing systems*. 2009, pp. 545–552.

[24] Zellig S. Harris. "Distributional structure." In: *Word* 10.2-3 (1954), pp. 146–162.

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

[26] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen." In: *Diploma, Technische Universität München* 91 (1991).

[27] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[28] Hakan Inan, Khashayar Khosravi, and Richard Socher. "Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling." In: *arXiv preprint arXiv:1611.01462* (2016).

[29] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. "Exploring the limits of language modeling." In: *arXiv preprint arXiv:1602.02410* (2016).

[30] Dan Jurafsky and James H. Martin. *Speech and language processing*. Vol. 3. Pearson London, 2014.

[31] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014).

[32] Reinhard Kneser and Hermann Ney. "Improved backing-off for m-gram language modeling." In: *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*. Vol. 1. IEEE. 1995, pp. 181–184.

[33] Jens Kober, J Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey." In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[34] Igor Kononenko. "Machine learning for medical diagnosis: history, state of the art and perspective." In: *Artificial Intelligence in medicine* 23.1 (2001), pp. 89–109.

[35] Siwei Lai, Kang Liu, Shizhu He, and Jun Zhao. "How to generate a good word embedding." In: *IEEE Intelligent Systems* 31.6 (2016), pp. 5–14.

[36] Tsungnan Lin, Bill G. Horne, Peter Tino, and C. Lee Giles. "Learning long-term dependencies in NARX recurrent neural networks." In: *IEEE Transactions on Neural Networks* 7.6 (1996), pp. 1329–1338.

[37] Yun Liu, Krishna Gadepalli, Mohammad Norouzi, George E Dahl, Timo Kohlberger, Aleksey Boyko, Subhashini Venugopalan, Aleksei Timofeev, Philip Q Nelson, Greg S Corrado, et al. "Detecting cancer metastases on gigapixel pathology images." In: *arXiv preprint arXiv:1703.02442* (2017).

[38] Thang Luong, Richard Socher, and Christopher D. Manning. "Better word representations with recursive neural networks for morphology." In: *CoNLL*. 2013, pp. 104–113.

[39] Gábor Melis, Chris Dyer, and Phil Blunsom. "On the state of the art of evaluation in neural language models." In: *arXiv preprint arXiv:1707.05589* (2017).

[40] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. "Regularizing and optimizing LSTM language models." In: *arXiv preprint arXiv:1708.02182* (2017).

[41] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. "Pointer sentinel mixture models." In: *arXiv preprint arXiv:1609.07843* (2016).

[42] Risto Miikkulainen and Michael G Dyer. "Natural language processing with modular PDP networks and distributed lexicon." In: *Cognitive Science* 15.3 (1991), pp. 343–399.

[43] Tomas Mikolov. "Statistical Language Models Based on Neural Networks." PhD thesis. 2012, pp. 1–129.

[44] Tomas Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur. "Recurrent Neural Network based Language Model." In: *Interspeech* September (2010), pp. 1045–1048.

[45] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. "Distributed Representations of Words and Phrases and their Compositionality." In: *Advances in neural information processing systems* (2013), pp. 3111–3119.

[46] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." In: *arXiv preprint arXiv:1301.3781* (2013).

[47] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning." In: *IEEE Transactions on knowledge and data engineering* 22.10 (2010), pp. 1345–1359.

[48] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. "How to Construct Deep Recurrent Neural Networks." In: March 2014 (2013). arXiv: `1312.6026`. URL: `http://arxiv.org/abs/1312.6026`.

[49] Jeffrey Pennington, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[50] Anna-Lena Popkes. "Representation Learning with Feedforward Neural Networks." Bachelor's Thesis. University of Osnabrück, August 2015.

[51] Ofir Press and Lior Wolf. "Using the output embedding to improve language models." In: *arXiv preprint arXiv:1608.05859* (2016).

[52] Radiology. *Oxford Dictionary*. https://en.oxforddictionaries.com/definition/radiology. Oxford University Press, 2018.

[53] Lev Ratinov and Dan Roth. "Design challenges and misconceptions in named entity recognition." In: *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics. 2009, pp. 147–155.

[54] Bruce Reiner and Eliot Siegel. "Radiology reporting: returning to our image-centric roots." In: *American Journal of Roentgenology* 187.5 (2006), pp. 1151–1155.

[55] Ronald Rosenfeld. "Two decades of statistical language modeling: Where do we go from here?" In: *Proceedings of the IEEE* 88.8 (2000), pp. 1270–1278.

[56] David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, et al. "Learning representations by back-propagating errors." In: *Cognitive modeling* 5.3 (1988), p. 1.

[57] Hacsim Sak, Andrew Senior, and Françoise Beaufays. "Long short-term memory recurrent neural network architectures for large scale acoustic modeling." In: *Fifteenth Annual Conference of the International Speech Communication Association*. 2014.

[58] Holger Schwenk and Jean-Luc Gauvain. "Training neural network language models on very large corpora." In: *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2005, pp. 201–208.

[59] Leslie N. Smith. "Cyclical learning rates for training neural networks." In: *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*. IEEE. 2017, pp. 464–472.

[60] Richard Socher. *Stanford University: Deep Learning for Natural Language Processing, Lecture 2: Word Vectors (min. 46-48)*. Youtube. March 2016. URL: https://www.youtube.com/watch?v=aRqn8t1hLxs (visited on 01/15/2018).

[61] Richard Socher, Cliff C. Lin, Chris Manning, and Andrew Y. Ng. "Parsing natural scenes and natural language with recursive neural networks." In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011, pp. 129–136.

[62] Harini Suresh, Nathan Hunt, Alistair Johnson, Leo Anthony Celi, Peter Szolovits, and Marzyeh Ghassemi. "Clinical Intervention Prediction and Understanding with Deep Neural Networks." In: *Machine Learning for Healthcare Conference*. 2017, pp. 322–337.

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

[64] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. "On the importance of initialization and momentum in deep learning." In: *International conference on machine learning*. 2013, pp. 1139–1147.

[65] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

[66] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. "Deep neural networks for object detection." In: *Advances in neural information processing systems*. 2013, pp. 2553–2561.

[67] Bo-Hsiang Tseng, Hung-yi Lee, and Lin-Shan Lee. "Personalizing universal recurrent neural network language model with user characteristic features by social network crowdsourcing." In: *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*. IEEE. 2015, pp. 84–91.

[68] Peter D. Turney and Patrick Pantel. "From frequency to meaning: Vector space models of semantics." In: *Journal of artificial intelligence research* 37 (2010), pp. 141–188.

[69] Tian Wang and Kyunghyun Cho. "Larger-Context Language Modelling." In: (2015), pp. 1–14. arXiv: `1511.03729`. URL: `http://arxiv.org/abs/1511.03729`.

[70] Will Williams, Niranjani Prasad, David Mrva, Tom Ash, and Tony Robinson. "Scaling Recurrent Neural Network Language Models." In: (2015), pp. 2–6. arXiv: `1502.00512`. URL: `http://arxiv.org/abs/1502.00512`.

[71] Yonghui Wu et al. "Google's neural machine translation system: Bridging the gap between human and machine translation." In: *arXiv preprint arXiv:1609.08144* (2016).

[72] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. "Show, attend and tell: Neural image caption generation with visual attention." In: *International Conference on Machine Learning*. 2015, pp. 2048–2057.

[73] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. "Breaking the Softmax Bottleneck: A High-Rank RNN Language Model." In: *arXiv preprint arXiv:1711.03953* (2017).

[74] Seunghyun Yoon, Hyeongu Yun, Yuna Kim, Gyu-tae Park, and Kyomin Jung. "Efficient Transfer Learning Schemes for Personalized Language Modeling using Recurrent Neural Network." In: *arXiv preprint arXiv:1701.03578* (2017).

[75] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. “Recurrent neural network regularization.” In: *arXiv preprint arXiv:1409.2329* (2014).

[76] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. “Recurrent highway networks.” In: *arXiv preprint arXiv:1607.03474* (2016).

[77] Barret Zoph, Deniz Yuret, Jonathan May, and Kevin Knight. “Transfer learning for low-resource neural machine translation.” In: *arXiv preprint arXiv:1604.02201* (2016).

[78] *spaCy - Industrial-Strength Natural Language Processing*. `http://dorienherremans.com/dlm2017/`.

EIDESSTATTLICHE ERKLÄRUNG