Karlsruhe Institute of Technology

Letter N-Gram-based Input Encoding for Continuous Space Language Models

Diploma Thesis of

Henning Sperr

At the Department of Informatics
Institute for Anthropomatics

Reviewer:	Prof. Dr. Alexander Waibel
Second reviewer:	Dr. Sebastian Stüker
Advisor:	Dipl. Inform. Jan Niehues

Duration: 01 January 2013 – 30 June 2013

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

Karlsruhe, 21.06.2013

. . . .

(**Sperr, Henning**)

Zusammenfassung

In dieser Arbeit präsentieren wir eine neue Eingabeschicht für Neuronale Netzwerk Sprachmodelle, sowie einen Entwurf für ein *Deep Belief Network*-Sprachmodell. Ein großes Problem der normalerweise genutzten Wortindex-Eingabeschicht ist der Umgang mit Wörtern, die zuvor in den Trainingsdaten nicht vorkamen. Bei manchen Neuronale Netzwerk Architekturen muss die Größe des Eingabevokabulars auf Grund der Komplexität und Rechenzeit eingeschränkt und somit die zusätzliche Menge der unbekannten Wörter vergrößert werden. Diese Netzwerke müssen auf ein normales N-Gram Sprachmodell zurückgreifen um dennoch eine Wahrscheinlichkeit zu liefern. Bei einer anderen Methode repräsentiert ein Neuron in der Eingabeschicht alle ungesehenen Wörter.

In der neuen *Letter N-Gram* Eingabeschicht wird dieses Problem behoben, indem wir Worte als Teilworte repräsentieren. Mit diesem Ansatz haben Wörter mit ähnlichen Buchstaben auch ähnliche Repräsentationen in der Eingabeschicht des Neuronalen Netzwerks. Wir wollen zeigen, dass es damit möglich ist für unbekannte Wörter dennoch sinnvolle Schätzungen abzugeben sowie morphologische Informationen eines Wortes zu erfassen, da diese Formen eines Wortes dennoch häufig die selben Buchstabenfolgen enthalten. Wir zeigen die Auswirkung im Bereich der statistischen maschinellen Übersetzung mithilfe eines Neuronalen Netzwerkes basierend auf *restricted Boltzmann machines*.

Wir benutzen diese Boltzmann-Maschinenarchitektur ebenfalls um die Gewichte des Deep Belief Networks zuvor zu trainieren und diese danach mit dem *Contrastive Wake Sleep*-Algorithmus abzustimmen. Für dieses Deep Belief Network haben wir eine Architektur gewählt, die vorher bereits bei handgeschriebener Ziffernerkennung gute Resultate erzielen konnte.

Wir untersuchen die Übersetzungsqualität sowie die Trainingszeit und unbekannte Wortrate bei einer Deutsch nach Englisch Übersetzungsaufgabe, welche die TED Daten benutzt. Wir benutzen die selben Systeme und untersuchen wie gut sie auf eine anderen domänenverwandten Aufgabe, nämlich Universitätsvorlesungen, generalisieren. Außerdem haben wir Experimente vom Englisch nach Deutsch auf Nachrichtentexten durchgefürt. Um die Systeme zu vergleichen benutzen wir die BLEU Metrik. Wir zeigen, dass mit der neuen Eingabeschicht eine Verbesserung von bis zu 1.1 Punkten erzielt werden kann. Die Deep Belief Network Architektur war in der Lage das Vergleichssystem ohne zusätzliches Sprachmodell um bis zu 0.88 Punkte zu verbessern.

Abstract

In this thesis we present an input layer for continuous space language models, as well as a layout for a deep belief network language model. A big problem of the commonly used word index input layer is the handling of words that have never been seen in the training data. There are also neural network layouts where the vocabulary size has to be limited due to the problem of complexity and computation time. These networks have to back-off to normal n-gram language models or use a neuron for all unknown words. With our new letter n-gram input layer this problem can be solved by modeling each word as a combination of subword features. With this approach words containing similar letter combinations will have close representations. We think this will improve the out-of-vocabulary rate as well as capturing morphological information about the words, since related words usually contain similar letters. We show their influence in the task of machine translation using a continuous space language model based on restricted Boltzmann machines.

We also used this Boltzmann machine architecture to pretrain the weights for a deep belief network and afterwards fine-tuning the system using the contrastive wake-sleep algorithm. For our deep belief network we chose a layout that proved to be able to do handwritten digit recognition. We evaluate the translation quality as well as the training time and out-of-vocabulary rates on a German-to-English translation task of TED and university lectures as well as on the news translation task translating from English-to-German. The systems were compared according to their BLEU scores. Using our new input layer approach a gain in BLEU score by up to 1.1 BLEU points can be achieved. The deep belief network architecture was able to improve the baseline system by about 0.88 BLEU points.

Contents

1. Introduction

In today's society cultural exchange becomes more important every day. The means of transportation and communication developed very fast over the last sixty years, so that now everyone is able to access huge amounts of information over the Internet or by travelling abroad. In this more and more globalised world the cultural and social borders of different countries grow closer together and make it necessary to speak or understand multiple languages.
For example in politics where communication between countries is crucial to maintain peace and understanding. If the translators provide false or bad translations the relation between two countries could become tense. In the European Parliament there are simultaneous translations for each nation. Everyone can hold a speech in his mother tongue and the audience can listen to the simultaneous translations in their mother tongue. This is also a very stressful task for humans since simultaneous translations require big effort and concentration and usually people work in teams and change shift in regular intervals.

Another area where multilingualism gains importance is the workplace, where often teams contain people from all over the world. Good communication is necessary to have a well functioning and productive work environment. Often English is considered to be the common spoken language not all people speak or understand it. Especially when traveling to different countries people in rural areas are not necessarily able to speak English. It is also not possible for everyone to learn three or more languages so there is an increasing need for translations. Professional human translation is expensive and sometimes not possible, especially for languages with very few speakers. That is why machine translation is an important field of research.

Computers are all around us in our daily life and being always connected to the Internet becomes more and more common. With the recent increase in smartphones, people are able to use machine translation software everywhere[1] [2]. This has many private applications, for example being able to ask for things or order in a restaurant during travel. But also in political area, where for example U.S. soldiers on a mission could translate from and to Arabic, to talk to local people during a mission. Another important task for mobile machine translation is for doctors that go abroad in third world countries to help, but do not speak the local languages. It is important to have good communication so the doctor is able to find the best way to cure the patient.

[1]http://www.jibbigo.com/website/
[2]http://www.google.de/mobile/translate/

Another important task is to make knowledge available to everyone. To do this, projects like Wikipedia or TED try to accumulate information, in many different languages, that is freely available for everybody. There is a big need, especially for science and research, to share results and knowledge through the Internet and across language borders. The number of universities that provide open lectures or video lectures online is growing and with this the need to automatically translate these lectures into different languages.

State of the art machine translation systems are good at doing domain limited tasks for close languages e.g. English-French, the performance is much worse for broad domain and very different languages e.g. English-Japanese. But even for close languages the translation quality for broad domain translation systems is still not on a professional level.

1.1. Language Modeling

State of the art systems use phrase-based statistical machine translation (SMT). The idea is to take parallel data to train the most probable translations from one language into the other. Phrases which consist of one or more words are extracted from an alignment, which usually is generated by taking the most probable word alignments from all the parallel sentences. These phrases are then stored in a phrase table and during decoding time used to translate chunks from the source language into the target language. This is a hard task since the number of possible translations grows exponentially.

The components used by such an SMT system can be described using the Bayes-Theorem which is

$$P(e|f) = \frac{P(f|e) \cdot P(e)}{P(f)} \tag{1.1}$$

Where $P(e|f)$ is the probability that sentence e is the translation for the given sentence f. This equals $P(f|e)$ the likelihood function, which is the probability of the sentence given the translation hypothesis. The probability of the translated sentence $P(e)$, is how likely it is that the translation hypothesis is a good sentence of the target language. This is divided by $P(f)$, the probability of the source sentence, which is usually omitted since it is constant for every sentence. The goal now is to find a sentence e so that this probability $P(e|f)$ is maximized rewriting above formula as

$$\arg\max_e P(e|f) = \arg\max_e P(f|e) \cdot P(e) \tag{1.2}$$

By doing so, the interesting components for optimization are the decoding process which equals finding the e with the maximum probability, the translation model which assigns the probability $P(f|e)$ to sentence pairs and the language model $P(e)$.

It is important to have a language model, since words can have many different translations in the target language and there needs to be a way to decide which is the best translation given a certain context. If only the translation model probabilities would be used for example, the most probable English translation for the German word “Bank” is “bank”. It would never be translated into “bench” which is another valid translation, depending on text context. Phrase-based translation compared to word-based translation helps solving this problem because words can be translated in context if the phrase has been seen during training, but this is not able to solve the problem entirely. Even with phrases there is still have no context information in between the phrase boundaries. The translation hypothesis might be very influent. By training a model that can assign a probability to a sentence, or parts of a sentence, a better estimate for translations using different contextual information can be obtained. During decoding time different pruning methods are used, due to the exponential growth of the search tree. Often, very improbable hypothesis will be cut off and a language model is one of the features that helps to evaluate whether the current hypothesis can be cut off or should be kept. Since the probability of a whole sentence, $P(W)$ is hard to estimate, because most of the sentences appear only once in the training data, a different approach is used. This approach is called n-gram-based language model, where the probability of a word given a limited history of words is calculated with

$$P(w_1^n) = P(w_1) \prod_{i=2}^{n} P(w_i | w_1^{i-1}) \tag{1.3}$$

Where w_1^i is a word sequence $w_1, ..., w_i$ and w_i is the i-th word. A good statistical language model should assign a non-zero probability to all possible word strings w_1^n but in practice there are several problems with the typical n-gram-based approach. One is that if the parameter N is too big, data sparseness will be a problem and there will be lots of n-grams that have no or very little statistical data. This means that different back-off or smoothing techniques have to be used to get a nonzero probability for those n-grams. If N is chosen too small, not enough context is captured to make meaningful distinctions. For example by using only unigram probabilities the model will give no information about fluency or translations in context. To deal with the data sparseness problems several approaches have been developed to calculate probabilities for unseen events. One example would be backing off to lower order n-grams [Kat87] under certain circumstances. The most common used technique today is called modified Kneser-Ney smoothing [KN95, CG96]. It is essentially a back-off interpolation algorithm with modified unigram probabilities.

In these standard back-off n-gram language models the words are represented in a discrete space, which prevents true interpolation of the probabilities for the unseen n-grams. This means that if the position in our discrete word space is changed it will lead to a discontinuous change in the n-gram probability. In this work we try to explore the use of continuous space language models (See Bengio et al. [BDVJ03]) to get a better approximate for n-grams that were never seen before. For this we developed new input layers for the restricted Boltzmann machine Language Model (RBMLM) of Niehues and Waibel [NW12] that try to capture morphological information of words by using different subword features. We try to overcome the problem of out-of-vocabulary (OOV) words by using subword features that will yield a good indicator for words that have the same function inside the sentence. The subword feature approach has the advantage that for big vocabularies the input layer become smaller and with that the training time of the network decreases.

Another part ot this work will be the exploration, whether so called Deep Belief Networks (DBNs), which proved to be good as acoustic models in speech recognition and digit image recognition [HDY$^+$12, rMDH12], are also useful in inferring the most probable n-gram. In the handwritten digit image recognition task (MINST) deep models achieved improvements over state of the art support vector machines in classifying different digits. In the speech recognition task a similar layout was used to project the cepstral coefficients of a context of frames onto the continuous space and use this vector to predict the hidden Markov model state. In our work, we try to project the history of a word through multiple layers and see if this higher order representation can model additional information that is useful to predict the probability of the current word. For this we took the design that Hinton et al. [HOT06] used for digit recognition on the MINST handwritten digit database.

1.2. Overview

The rest of this thesis is structured as follows:

In Chapter 2 we will give an overview of the field of neural networks and language modeling using continuous space representations. After discussing different approaches to the topic we want to contrast the work done in this thesis with work that has been done before.

In Chapter 3 the basics of neural networks that are needed to understand the work done in this thesis will be presented. We will explain basic feed forward neural networks and the difference to Boltzmann machines. We will explain how restricted Boltzmann machines are used to pretrain deep neural networks and introduce an algorithm that will be used for the fine-tuning of this model.

In Chapter 4 we will introduce two basic continuous space language modeling approaches and how they try to infer probabilities for a word n-gram. We will also explain in case of the restricted Boltzmann machine, how the 1-in-n coding word index works.

In Chapter 5 we will motivate the proposed input layer and introduce additional features that can be used with the new input layer.

In Chapter 6 we will give a brief introduction to the deep belief network language model layout that we propose in this thesis. We will explain on why we decided to use the given layout and explain how the probabilities are going to be inferred from this network.

In Chapter 7 we will talk about the design of the implementation and explain a few methods that helped while debugging the code and how to visually check if learning is working well or not.

In Chapter 8 we will explain on which systems we tested our models and describe the outcome of different experiments. We will explore if the input layer proposed in this work achieves better performance then earlier proposed methods and do experiments on different improvements of the input layer.

In Chapter 9 we will conclude our work and give an outlook over future work that can be done to further enhance this research.

2. Previous Work

Research about neural networks in natural language processing dates back into the 1990s where for example Waibel et al. [WHH^{+}90] used time delay neural networks to improve speech recognition quality. Gallant [Gal91] used neural networks to resolve word sense disambiguations. Neural networks were also used as part-of-speech tagger in Schmid [Sch94a]. One of the first approaches to use them in language modeling was in Nakamura and Shikano [NMKS90], where a neural network was used to predict word categories. Miikulainen and Dyer researched another approach to continuous natural language processing based on hierarchically organized subnetworks with a central lexicon [MD91].

At that time the vast computational resources were not available so that, after a short recession, now in recent times new neural network approaches have come up again. Xu and Rudnicky [XR00] proposed a language model that has an input consisting of one word and no hidden units. This network was limited to infer unigram and bigram statistics. One of the first continuous space language models that paved the way for methods that are used today was presented in Bengio et al. [BDVJ03]. They used a feed forward multilayer perceptron as n-gram language model and achieved a decrease in perplexity between 10 and 20 percent. In Schwenk and Gauvain [SG05] and later in Schwenk [Sch07] research was performed on training large scale neural network language models on millions of words resulting in a decrease of the word error rate for continuous speech recognition. In Schwenk [Sch10] they use the continuous space language modeling framework to re-score n-best lists of a machine translation system during tuning. Usually these networks use short lists to reduce the size of the output layer and to make calculation feasible. There have been approaches to optimize the output layer of such a network, so that vocabularies of arbitrary size can be used and there is no need to back off to a smaller n-gram model Le et al. [LOM^{+}11]. In this Structured Output Layer (SOUL) neural network language model a hierarchical output layer was chosen. Recurrent neural networks have also been used to try and improve language model perplexities in Mikolov et al. [MKB^{+}10], concluding that Recurrent neural networks potentially improve over classical n-gram language models with increasing data and a big enough hidden unit size of the model. In the work of Mnih and Hinton [MH07] and Mnih [Mni10] training factored restricted Boltzmann machines yielded no gain compared to Kneser-Ney smoothed n-gram models. But it has been shown in Niehues and Waibel [NW12], that using a restricted Boltzmann machine with a different layout during decoding can yield an increase in BLEU score.

There has also been research in the field of using subword units for language modeling. In Shaik et al. [SMSN11] linguistically motivated sub-lexical units were proposed to improve open vocabulary speech recognition for German. Research on morphology-based and subword language models on a Turkish speech recognition task has been done by Sak et al. [SSG10]. The idea for Factored Language models in machine translation has been introduced by Kirchhoff and Yang [KY05]. Similar approaches to develop joint language models for morphologically rich languages in machine translation have been presented by Sarikaya and Deng [SD07].In Emami et al. [EZM08] a factored neural network language model for Arabic language was built. They used different features such as segmentation, part-of-speech and diacritics to enrich the information for each word. We are also going to investigate, if a deep belief network trained like described in Hinton et al. [HOT06] can get a similar good performance as in image recognition and acoustic modeling. For this we use Boltzmann machines to do layer by layer pretraining of a deep belief network and after that wake-sleep fine-tuning to find the final parameters of the network. To the best of our knowledge using a restricted Boltzmann machine with subword input vectors to capture morphology information is a new approach and has never been done before. The approach to project words on a continuous space through multiple layers and use the top level associative memory to sample the free energy of a word given the continuous space vector has as far as we know not been done yet.

3. Neural Networks

A neural network is a mathematical model, which tries to represent data in the same way the human brain does. It consists of neurons that are connected with each other and using learning techniques the connections between different neurons strengthen or weaken. The weights of these connections determine how much a neuron that activates influences the neurons that are connected to it. It is often used to model complex relations and find patterns in data.

Neural networks were originally developed in 1943 by Warren McCulloch and Walter Pitts. They elaborated on how neurons might work and they modeled a simple network with electrical circuits. In 1949 Donald Hebb pointed out in his book ”The Organization of Behavior” that neural pathways that are often used get stronger, which is a crucial part to understand human learning. During the 1950's with the rise of von Neumann computers the interest in neural networks decreased until in 1958 Frank Rosenblatt came up with the perceptron, which led to a reemerge of the field. Soon after that in 1969 Minsky and Papert wrote a book on the limitations of the perceptron and multilayer systems, which led to a significant shortage in funding for the field. In 1974 Paul Werbos developed the back-propagation learning method, which was simultaneously rediscovered in 1985-1986 by Rumelhart and McClelland and is now probably the most well known and widely used form of learning today.

Recently neural networks are regaining attention since the advancement in information technology and the development of new methods makes training deep neural networks feasible. The term *deep belief network* describes a generative model that is composed of multiple hidden layers. It is possible to have many different layouts of the neurons and connections in such a network and one of the most common layouts is similar to the feed forward network (See Figure 3.1). In a feed forward network we have an input layer representing the data we want to classify. After that we can have one or multiple hidden layers until we reach the output layer which will contain the feature vector for the given input data. Another common layout is the recurrent neural network in which the connections of the units form a directed circle. This allows to capture dynamic temporal behavior.

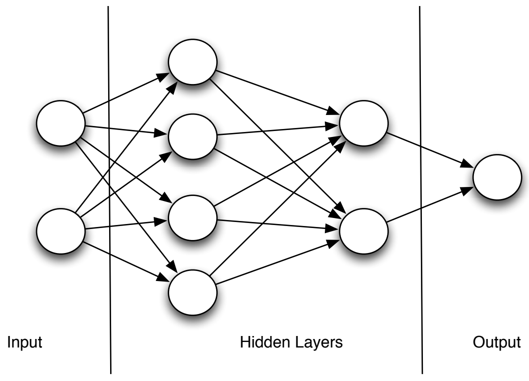

Figure 3.1.: Example for a feed forward neural network.

3.1. Feed Forward Neural Networks

A typical feed forward network consists of an input layer, one or many hidden layers and an output layer. The decision if a neuron turns on or off is usually chosen by summing the input of the neurons in the layer below, which is either on or off for binary neurons multiplied by the weight of the connection and a *sigmoid* activation function (Figure 3.2). This way the data is represented by the input neurons and propagated through the weights into the hidden layer until the output layer. The logistic sigmoid function is defined as

$$p(N_i = on) = \frac{1}{1 + \mathrm{e}^{-t}} \tag{3.1}$$

where N_i is the ith neuron and t is the input from other neurons. Usually a threshold is set to decide whether the neuron turns on or off. In some training algorithms though, a sampling strategy is used to sometimes accept improbable samples (e.g. a high probability neuron not turning on) to get a random walk over the trainings surface and not get stuck in local optima.

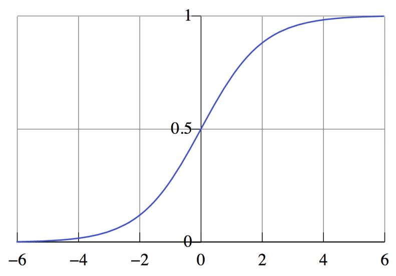

Figure 3.2.: Sigmoid activity function.[1]

3.1.1. Training

Training a neural network is a difficult task. There are three major paradigms for the learning task.

[1]Source: `http://en.wikipedia.org/wiki/Sigmoid_function`

1. In *Supervised Learning* we have a set of data and the corresponding classes and we try to change the parameters of the model in a way that the data gets classified correctly. A cost function is used to determine the error between correct labels and network output.
2. In *Unsupervised Learning* we have only data without labels and need to define a cost function that measures the error given the data and the mapping of our network.
3. In *Reinforcement Learning* we think of the problem as a Markov decision process, where we have several states and in each of these states we have probabilities of going into one of the other states. For each of these transitions a reward or punishment is given according to the output. In this way we try to learn the best transition weights and probabilities to get a good model.

The most commonly used algorithm is a supervised learning method called back-propagation. In this algorithm examples are shown to the network and the output of the neural network is compared with the expected output. The motivation for this algorithm is that a child learns to recognize things by seeing or hearing many examples an getting "labels" from other people who tell the child what it is seeing. Having enough data and labels shallow neural networks can be trained very well using this algorithm. Commonly the weights are initialized randomly and trained for some iterations. The big problem with random weights and deep networks is that back-propagation tends to settle in very bad local minima. There are different approaches to cope with this problem, for example using Simulated Annealing techniques. In this work we will not initialize the weights randomly but use a restricted Boltzmann machine to pretrain the weights of a deep belief network. Another problem in training deep networks with back-propagation is a phenomenon called *explaining away.* This will be further described in the Chapter 6.1.

3.2. Boltzmann Machines

A Boltzmann machine is a type of stochastic recurrent neural network in which there are a number of neurons that have undirected symmetric connections with every other neuron. Such a network with five neurons can be seen in Figure 3.3.

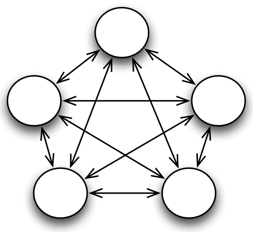

Figure 3.3.: Example for a Boltzmann machine having five neurons.

Since learning in a normal Boltzmann machine with unrestricted connections is difficult and time consuming often restricted Boltzmann machines are used instead.

3.2.1. Restricted Boltzmann Machines

In a restricted Boltzmann machine an additional constraint is imposed on the model. We have no connections in the hidden and visible layer. This means that a visible node has only connections to hidden nodes and hidden nodes only to visible nodes (Figure 3.4)

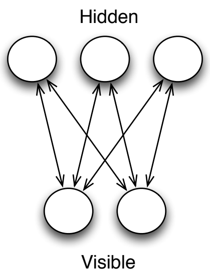

Figure 3.4.: Example for a restricted Boltzmann machine with three hidden units and two visible units.

The restricted Boltzmann machine is an energy-based model that associates an energy to each configuration of neuron states. Through training we try to give good configurations low energy and bad configurations high energy. Since we try to infer ”hidden” features in our data we will have some neurons that correspond to visible inputs and some neurons that correspond to hidden features. We try to learn good weights and biases for these hidden neurons during training and we hope that they are able to capture higher level data.

The Boltzmann machine assigns the energy according to the following formula

$$E(v,h) = -\sum_{i \in \text{visible}} a_i v_i - \sum_{j \in \text{hidden}} b_j h_j - \sum_{(i,j) \in (\text{visible},\text{hidden})} v_i h_j w_{ij} \tag{3.2}$$

where v and h are visible and hidden configurations and $w_{i,j}$ is the weight between unit i and j. Given this Energy Function we assign the probability in such a network as

$$p(v,h) = \frac{1}{Z}\, \mathrm{e}^{-E(v,h)} \tag{3.3}$$

with the partition function Z as

$$Z = \sum_{v,h} \mathrm{e}^{-E(v,h)} \tag{3.4}$$

where v and h are visible and hidden configurations. Since we are interested in the probability of our visible input we can marginalize Equation 3.3 over the hidden units and get

$$p(v) = \frac{1}{Z} \sum_{h} \mathrm{e}^{-E(v,h)}\,. \tag{3.5}$$

Usually the partition function Z is not feasible to compute since it is a sum over all possible visible and hidden configurations. To overcome this problem we will not use the probability of our visible input but something called free energy. The free energy of a visible configuration in a restricted Boltzmann machine with binary stochastic hidden units is defined as

$$F(v) = -\sum_i v_i a_i - \sum_j log(1 + \mathrm{e}^{x_j}) \tag{3.6}$$

with $x_j = b_j + \sum_i v_i w_{ij}$. It can also be shown that

$$\mathrm{e}^{-F(v)} = \sum_h \mathrm{e}^{-E(v,h)} \,. \tag{3.7}$$

Inserting this into Equation 3.5 we get

$$p(v) = \frac{1}{Z} \mathrm{e}^{-F(v)} \,. \tag{3.8}$$

This formula still contains the partition function but we also know that the free energy will be proportional to the true probability of our visible vector.

3.2.2. Contrastive Divergence Learning

For this work we used Contrastive Divergence learning as proposed in Hinton [Hin02]. It is an unsupervised learning algorithm that uses Gibbs-Sampling. The implementation is based on Hinton [Hin10]. In order to do this, we need to calculate the derivation of the probability of the example given the weights:

$$\frac{\delta \log p(v)}{\delta w_{ij}} = <v_i h_j>_{\text{data}} - <v_i h_j>_{\text{model}} \tag{3.9}$$

where $<v_i h_j>_{\text{model}}$ is the expected value of $v_i h_j$ given the distribution of the model. In other words we calculate the expectation of how often v_i and h_j are activated together given the distribution of the data, minus the expectation of them being activated together given the distribution of the model, which will be calculated using Gibbs-Sampling techniques. The first part is easy to calculate since we just input the data and propagate the activations into the hidden units and count which of the pairs are on together and which are off. For the estimation of $<v_i h_j>_{\text{model}}$ usually many steps of Gibbs-Sampling are necessary to get an unbiased sample from the distribution. This can be seen in Figure 3.5 where the green units depict the activation of the data and the red units are the so called fantasy particles inferred from the distribution of the model. The common implementation of the Contrastive Divergence algorithm only performs one step of sampling. Minimizing this Contrastive Divergence function works well as shown in several experiments. [Hin02,CPH].

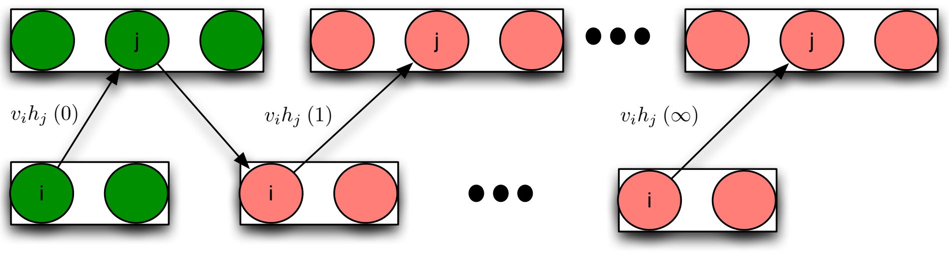

Figure 3.5.: Markov Chain of the Contrastive Divergence learning step.

3.3. Deep Neural Networks

In recent times deep neural network language models are beginning to gain importance, since they achieved good results in speech recognition and image recognition tasks. With increasing computational power training large neural networks with multiple hidden layers becomes more and more feasible. Despite that it is still hard to train many hidden layers of a directed belief network using back-propagation. Especially when the weights are randomly initialized the algorithm will easily get stuck in local optima. Inference in deep directed neuronal networks having many hidden layers is difficult, due to the problem of "explaining away". A common example[2] for the problem can be seen in Figure 3.6. The two independent causes for the jumping house become highly dependent when we see the house jumping. The weights of the arrows coming from the outside represent the bias and in case of the earthquake node -10 means that without any other input the node is e^{10} times more likely to be off then on. If we now observe that the earthquake node is on and the truck node is off the total input of the jumping house node is zero which means that the node will turn on half of the time. This is a much better explanation that the house jumped than the probability of e^{-20} we get when earthquake and truck node are off, which would mean the house jumped on its own. It is also unlikely to turn on both the earthquake and the truck node since the likelihood for that happening is also $\mathrm{e}^{-10} \cdot \mathrm{e}^{-10} = \mathrm{e}^{-20}$. If we now want to know the probability of the truck hitting the house, when we know the house jumps the probability changes whether the earthquake node is turned on or off. When the earthquake node is turned on it "explains away" the evidence for the truck hitting the house. This problem happens when we have several hidden unobserved neurons that influence each other.

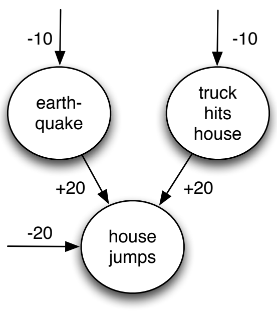

Figure 3.6.: Example for "explaining away" in a directed deep neuronal network.

There are Markov chain Monte Carlo methods [Nea92] which try to cope with this problem but they are usually very time consuming. In Hinton et al. [HOT06] they used complementary priors to eliminate the explaining away effect using a greedy layer wise training with restricted Boltzmann machines. In our work we will use a so called deep belief network (e.g. Figure 3.7), which is a probabilistic generative model. It is composed of an input layer and several hidden layers. These hidden layers are typically binary and the top two layers have undirected symmetric connections which form a so called associative memory.

[2]This example is taken from Hinton et al. [HOT06]

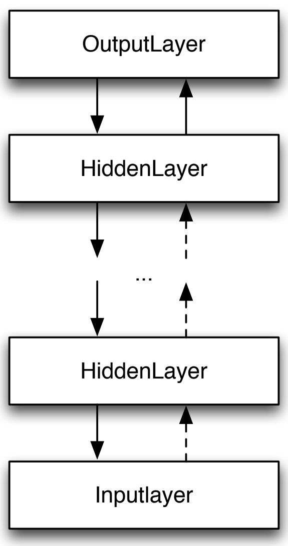

Figure 3.7.: Layout of a deep belief network.

3.3.1. Training

The training of a deep belief network will be greedy layer wise following these steps:

1. Train the first layer as an RBM that models the distribution $P(h^0|x)$ of the first hidden layer h^0 given the data x.
2. Use the first layer to obtain a representation of the input and use this as data for the second layer to calculate $P(h^1|h^0)$. We will use the real valued activations from the lower level hidden activities. We then use this data to train the next layer RBM.
3. We repeat 1. and 2. for the desired number of layers.
4. The last step can be fine-tuning the connections in between the layers since after pretraining we have a good initialization for algorithms like back-propagation or the wake-sleep algorithm

Each of the RBMs will be trained separately. If the number of weights of a preceding layer is the same as in the current layer we will initialize the weights from the layer below instead of using random values. When we pretrained each layer using the Contrastive Divergence algorithm, the weights of the layers already have a good initialization to classify the data. But since the layers were trained independently the transition from one layer to the next can be improved by using a fine-tuning algorithm. This can usually be back-propagation or in our case the contrastive wake-sleep algorithm as described in Hinton et al. [HOT06].

3.3.2. Contrastive Wake-Sleep Fine Tuning

Contrastive wake-sleep fine-tuning is an unsupervised algorithm to fine tune a deep belief network. For this algorithm the weights and biases of the network will be split up into generative weights (top-down weights) and recognition weights (bottom-up weights). The algorithm itself consists of two phases namely the wake and sleep phase. During the wake phase the data is propagated bottom up through the network where the binary feature vectors of one layer are used as data for the next. The top level associative memory layer is then trained with a few steps of Contrastive Divergence training. This wake phase can be seen as the model observing the real data. The Contrastive Divergence learning in the associative memory can be seen as the process of dreaming, where the reconstructions obtained using Gibbs sampling represent what the model likes the data to look like. This reconstructed vector is then propagated back down into the input layer in the so called sleep phase. These vectors obtained during the wake and sleep phase are used to train the recognition and generative weights in a way to better represent the data. For more information and a pseudo-code implementation of this algorithms please refer to Hinton et al. [HOT06].

4. Continuous Space Language Models

In this chapter we will take a brief look at different continuous space language model approaches that have been introduced in the past. We will introduce one feed forward language model approach and then compare it to a restricted Boltzmann machine language model. In this work we propose extensions to the Boltzmann machine model and also use it to pretrain a deep belief network as explained in Chapter 6.1.

4.1. Feed Forward Language Model

The CSLM toolkit of Holger Schwenk [Sch07, Sch10, SRA12] is a freely available toolkit to build a feed forward neural network language model. The download is available on the homepage[1]. In Figure 4.1 we can see the general layout of the CSLM Toolkit. The input of the network is the word history where every word is in the 1-of-N coding. First the words get projected onto a feature vector using shared weights between all words to do this projection into a continuous space. This projection layer uses a linear activation function. Then the continuous space vectors are propagated through a hidden layer using a hyperbolic tangent activation functions. The output layer contains a node for each word of the vocabulary. The activation of these neurons correspond to the probability $P(w_i|h_j)$ of the word w_i given the history h_j using:

$$p_i = \frac{\mathrm{e}^{x_i}}{\sum_{j=1}^{K} \mathrm{e}^{x_j}}. \tag{4.1}$$

where p_i is the probability of the word w_i given the current context and x_i is the sigmoid activation of the ith neuron. The network is trained using back-propagation.

[1]http://www-lium.univ-lemans.fr/cslm/

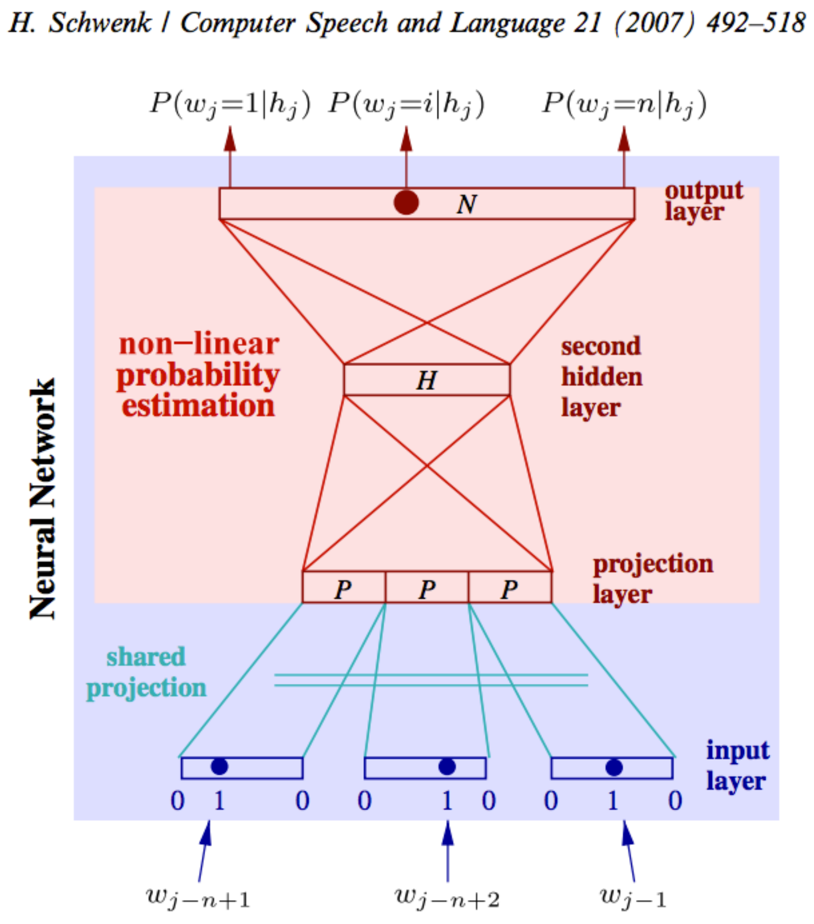

Figure 4.1.: Layout of the neural network used in the CSLM Toolkit.

The main problem of this approach is the high computational complexity. Calculating all output probabilities for each history query is far to expensive to be used in the decoding process. As we can see in Equation 4.1 we need to calculate them for the normalization of the output probability. However there have been methods to reduce the complexity of this model by using *short-lists*. With this approach the size of the output layer is reduced to, for example, the most probable 1024 words, decreasing the size of the output layer. If the probability for a word that is not contained in this short-list is needed, a back-off language model is used. Another method to speed up the calculation is called *Bunch mode*. In this approach multiple queries are simultaneously propagated through the network converting the vector/matrix operations to a matrix/matrix operation which helps parallelizing when using GPU computation and can also be heavily optimized on current CPU architectures using libraries like Intel MKL or BLAS. The third major optimization is called *grouping*, where calls for different words sharing the same history are grouped together, using the fact that we always calculate the probability for each word in the vocabulary given a history.

4.2. Restricted Boltzmann Machine Language Model

The word index restricted Boltzmann machine language model was developed by Niehues and Waibel [NW12] and is used as a baseline to test the proposed extensions of the language model. Words for each context are coded as a 1-of-N coding where each word is represented as a vector with one value set to one and the rest to zero. This is encoded in the network as a *softmax* visible layer for each context. *Softmax* means that the activation of the neurons gets restricted to one neuron at a time, since having more than one neuron activate would mean that there are two different words at the same time at the same position. The activation of a neuron is calculated as

$$p(v_i = \text{on}) = \frac{\mathrm{e}^{x_i}}{\sum_{j=1}^{K} \mathrm{e}^{x_j}}. \tag{4.2}$$

where v_i is the ith visible neuron and x_i is the input from the hidden units for the ith neuron defined as

$$x_i = a_i + \sum_j h_j w_{ij}. \tag{4.3}$$

with a_i being the bias of the visible neuron v_i and $h_j w_{ij}$ is the contribution of the hidden unit h_j to the input of v_i. When we now input a n-gram into the network we set the neuron representing the word position inside the vocabulary to *on* and all the others in the same layer to *off*. In Figure 4.2 is an example model with three hidden units, two contexts and a vocabulary of two words, in this example from the n-gram "my house".

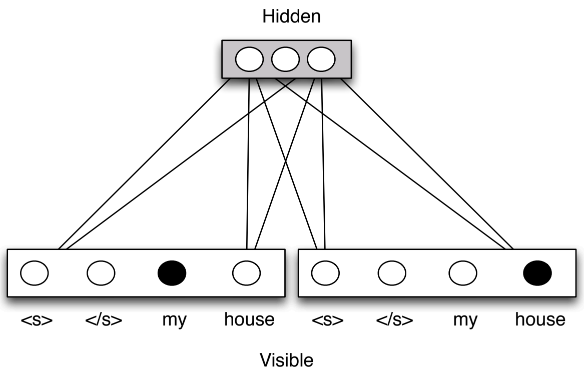

Figure 4.2.: A word index RBMLM with three hidden units and a vocabulary size of two words and two contexts.

Using this model we do not directly calculate the probability of an n-gram. Instead we calculate the free energy of a given word input v according to Equation 3.6. We will use this free energy as a feature in the log linear model of the decoder. In order to do this we need to be able to compute the probability for a whole sentence. As shown in Niehues and Waibel [NW12] we can do this by summing the free energy of all n-grams contained in the sentence. An advantage of the RBM model compared to the feed forward approach is that calculating the free energy as defined in Equation 3.6 only depends on the hidden layer size. Since the input vectors use the 1-of-N coding with only one neuron activated in each context, the free energy can be calculated very fast.

5. Letter-based Word Encoding

In this chapter we are going to introduce the newly proposed letter n-gram subword features. First we want to motivate the approach and then explain the concrete implementation of the letter n-gram subword features. After that we will explain the further modifications called capital letter and tree-split features. This input encoding can be used with any CSLM approaches, but in this work we will evaluate the approach using restricted Boltzmann machine language models.

5.1. Motivation

In the example mentioned above, the word index model might be able to predict *my house* but it will fail on *my houses* if the word *houses* is not in the training vocabulary. In this case, a neuron that classifies all unknown tokens or some other techniques to handle such a case have to be utilized. In contrast, a human will look at the single letters and see that these words are quite similar. He will most probably recognize that the appended *s* is used to mark the plural form, but both words refer to the same thing. He will be able to infer the meaning although he has never seen it before. Another example in English are be the words *happy* and *unhappy*. A human speaker who does not know the word *unhappy* will be able to know from the context what *unhappy* means and he can guess that both of the words are adjectives, that have to do with happiness, and that they can be used in the same way. In other languages with a richer morphology, like German, this problem is even more important. The German word *schön* (engl. *beautiful*) can have 16 different word forms, depending on case, number and gender. Humans are able to share information about words that are different only in some morphemes like *house* and *houses*. With our letter-based input encoding we want to generalize over the common word index model to capture morphological information about the words to make better predictions for unknown words.

5.2. Features

In order to model the aforementioned morphological word forms, we need to create features for every word that represent which letters are used in the word. If we look at the example of *house*, we need to model that the first letter is an *h*, the second is an *o* and so on.

If we want to encode a word this way, we have the problem that we do not have a fixed size of features, but the feature size depends on the length of the word. This is not possible in the framework of continuous space language models. Therefore, a different way to represent the word is needed.

An approach for having a fixed size of features is to just model which letters occur in the word. In this case, every input word is represented by a vector of dimension n, where n is the size of the alphabet in the text. Every symbol, that is used in the word is set to one and all the other features are zero. By using a sparse representation, which shows only the features that are activated, the word *house* would be represented by

$$w_1 = \text{e h o s u}$$

The main problem of this representation is that we lose all information about the order of the letters. It is no longer possible to see how the word ends and how the word starts. Furthermore, many words will be represented by the same feature vector. For example, in our case the words *house* and *houses* would be identical. In the case of *houses* and *house* this might not be bad, but the words *shortest* and *others* or *follow* and *wolf* will also map to the same input vector. These words have no real connection as they are different in meaning and part of speech.

Therefore, we need to improve this approach to find a better model for input words. N-grams of words or letters have been successfully used to model sequences of words or letters in language models. We extend our approach to model not only the letters that occur in the in the word, but the letter n-grams that occur in the word. This will of course increase the dimension of the feature space, but then we are able to model the order of the letters. In the example of *my house* the feature vector will look like

$$\begin{aligned} w_1 &= \text{my <w>m y</w>} \\ w_2 &= \text{ho ou se us <w>h e</w>} \end{aligned}$$

We added markers for the beginning and end of the word because this additional information is important to distinguish words. Using the example of the word *houses*, modeling directly that the last letter is an *s* could serve as an indication of a plural form.

If we use higher order n-grams, this will increase the information about the order of the letters. But these letter n-grams will also occur more rarely and therefore, the weights of these features in the RBM can no longer be estimated as reliably. To overcome this, we did not only use the n-grams of order n, but all n-grams of order n and smaller. In the last example, we will not only use the bigrams, but also the unigrams.

This means *my house* is actually represented as

$$\begin{aligned} w_1 &= \text{ m y my <w>m y</w>} \\ w_2 &= \text{e h o s u ho ou se us <w>h e</w>} \end{aligned}$$

With this we hope to capture many morphological variants of the word *house*. Now the representations of the words *house* and *houses* differ only in the ending and in an additional bigram.

$$\begin{aligned} \textit{houses} &= \text{... es s</w>} \\ \textit{house} &= \text{... se e</w>} \end{aligned}$$

The beginning letters of the two words will contribute to the same free energy only leaving the ending letter n-grams to contribute to the different usages of *houses* and *house*.

The actual layout of the model can be seen in Figure 5.1. For the sake of clarity we left out the unigram letters. In this representation we now do not use a softmax input layer, but a logistic input layer defined as

$$p(v_i = \text{on}) = \frac{1}{1 + \mathrm{e}^{-x_i}} \tag{5.1}$$

where v_i is the ith visible neuron and x_i is the input from the hidden units as defined in Section 4.2.

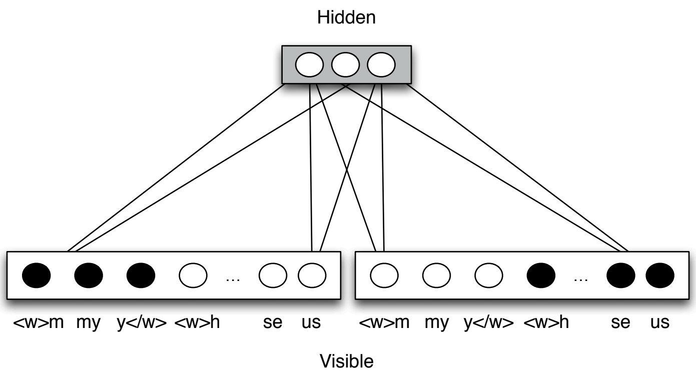

Figure 5.1.: A bigram letter index RBMLM with three hidden units.

5.2.1. Additional Information

The letter index approach can be extended by several features to include additional information about the words. This could for example be part-of-speech tags or other morphological information. In this work we tried to include a neuron to capture capital letter information. To do this we included a neuron that will be turned on if the first letter was capitalized and another neuron that will be turned on if the word was written in all capital letters. The word itself will be lowercased after we extracted this information.

Using the example of *European Union*, the new input vector will look like this

$$\begin{aligned} w_1 =& \text{a e n o p r u an ea eu op pe ro ur} \\ & \text{<w>e n</w><CAPS>} \\ w_2 =& \text{u i n o un io ni on} \\ & \text{<w>u n</w><CAPS>} \end{aligned}$$

This will lead to a smaller letter n-gram vocabulary since all the letter n-grams get lowercased. This also means there is more data for each of the letter n-gram neurons that were treated differently before. We also introduced an all caps feature which is turned on if the whole word was written in capital letters. We hope that this can help detect abbreviations which are usually written in all capital letters. For example *EU* will be represented as

$$w_1 = \text{e u eu <w>e u</w><ALLCAPS>}$$

5.2.2. Tree-Split Features

Another approach we tried was to split the word hierarchically and segment the letter n-grams depending on where inside of the word they appear. This can help model informations like *ed* or *ing* being a common ending for an English word but not as probable at the beginning or in the middle. This splitting was performed recursively up to a given tree depth. For each step we generate the letter n-gram of the current word or subword and add them to the final output.

In the case of the word *European* and a tree depth of one we generate the normal letter n-grams like above and then we split the word into *Euro* and *pean*. We then calculate the letter n-gram of each of these tokens and combine them with information about the tree layer and context these letter n-gram have been found. If we want to get a deeper split we would continue to split each of the two tokens in the middle and perform the same steps again. An example of this can be seen in Figure 5.2.

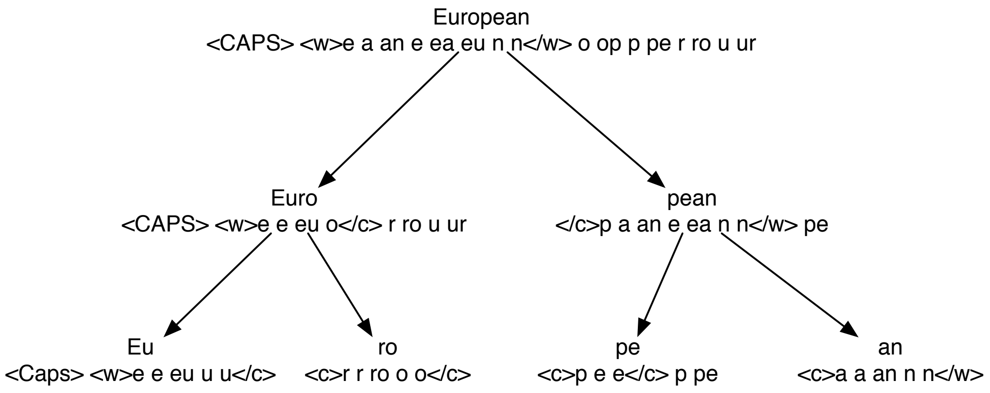
European
<CAPS> <w>e a an e ea eu n n</w> o op p pe r ro u ur
Euro
<CAPS> <w>e e eu o</c> r ro u ur
pean
</c>p a an e ea n n</w> pe
Eu
<Caps> <w>e e eu u u</c>
ro
<c>r r ro o o</c>
pe
<c>p e e</c> p pe
an
<c>a a an n n</w>

Figure 5.2.: Example of the tree splitting of input words with tree depth two and letter bigrams.

After we created the tree for all the words in our vocabulary we use the union of all tokens as new vocabulary. The new tokens are composed of the letter n-gram seen and the tree depth it was found in and the context in the current tree depth. This will determine the input layer of our network. We inserted <c> and </c> to mark the endings and beginnings of contexts in the lower layers of the tree. For the sake of visibility we will use the word *word* as an example and construct our final input layer vocabulary with a letter n-gram context of two and a tree depth of one. A tree containing the factores can be seen in Figure 5.3.

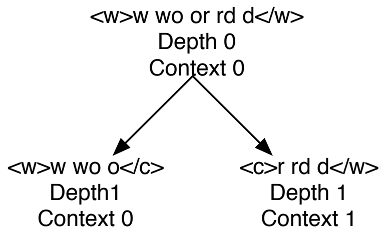

Figure 5.3.: Example of new tokens.

Using this tree the final tokens for each layer can be found in Table 5.1.

Depth	Word:Depth:Context
0	<w>w:d0:c0 wo:d0:c0 or:d0:c0 rd:d0:c0 d</w>:d0:c0
1	<w>w:d1:c0 wo:d1:c0 </c>:d1:c0 <c>r:d1:c1 rd:d1:c1 d<w>:d1:c1

Table 5.1.: Actual tokens for the word *word* generated by the tree-split algorithm.

We can see that each letter n-gram gets additional information about depth and context it was found in. This means that the *wo* token found in tree depth zero will be represented by a different neuron than the one found in tree depth one in the first context. Using the union of all these tokens will increase input vocabulary since we have neurons for each letter n-gram in each depth and context it has been found in. This example contains 11 tokens in total for one word. In a very small vocabulary with one or two words the vocabulary size will increase over the word approach but the more words are in the vocabulary the more tokens will repeat.

6. Deep Belief Network Language Model

In this chapter we want to explain the deep belief network language model. We used the same layout as Hinton et al [HOT06] used for image recognition. The layout can be seen in Figure 6.1.

First we take either the word indices as described in Section 4.2 or the newly proposed features described in Chapter 5 and split the n-gram in word and its history. The history of the word will then be propagated from the bottom layer through multiple hidden layers until we reach the layer connected to the hidden units in the associative memory. In this layer we combine the input with the word index or letter n-gram index to form the input for the associative memory. We then do a few steps of Contrastive Divergence sampling in the associative memory before we calculate the free energy. this top-level free energy will then be used as feature value for the log-linear model in the decoder.

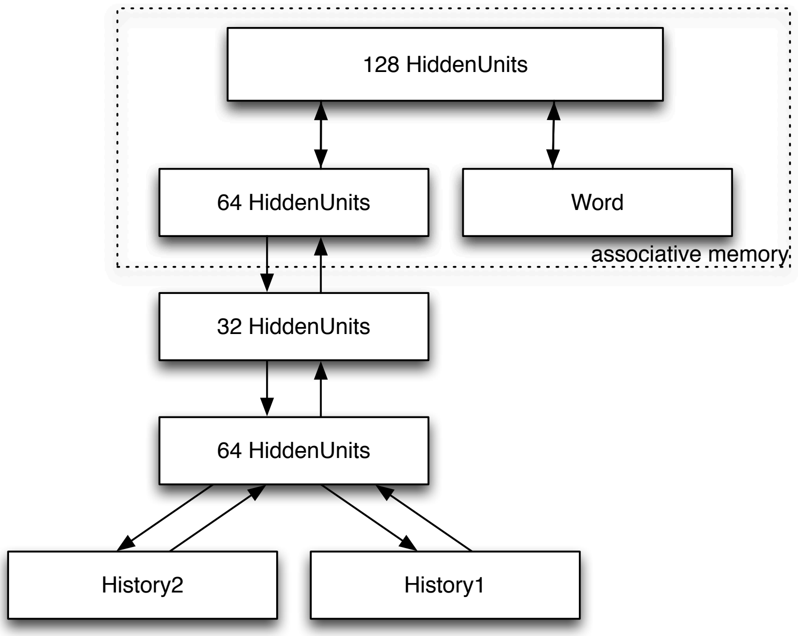

Figure 6.1.: Example layout of the deep belief network language modeling.

This layout was used in handwritten digit recognition, propagating the pixels of the image through the network and mixing it with the correct label as input in the top layer associative memory. We used a network with 8, 8, 8, 32 ad 16, 8, 16, 32 neurons. We chose the first layout using 8, 8, 8, 32 neurons because we wanted to see how propagating the history through equally sized layers will perform, since we use the trained weights of the lower layer to initialize the weights for the upper layer training. The second layout was chosen because sometimes it seems to be helpful to create a bottleneck in the network to force it to learn a compressed representation of the data [HS06].

7. Implementation

In this chapter we want to elaborate the design of a neural network software and software design in general since implementation was also a big part of this work. Since we try to produce software that is flexible and easy to maintain we try to implement software following the "clean code"[1] guidelines. The most fundamental principle is to develop code that is easy to read and understand, so that everyone who wants to work with the code can easily understand what it does. We also want to reduce so called "code smells" to make the code less prone to errors and flexible to change for later use. Although clean code is never a finished progress and code can always improve in clarity and readability we tried our best to design the interfaces and code in a way that tries to meet those coding standards.

7.1. Design

The design of the RBMLM framework is flexible to support different types of input layers and different types of data. We implemented simple interfaces and tried to get high re-usability. It is also easy to implement additional types of data for example mel cepstral coefficients as in Dahl et al. [DYDA12] or Hinton et al. [HDY$^+$12] or building a translation model similar to Schwenk [Sch12,SAY12]. A brief sketch of the layout can be seen in Figure 7.1 and we want to give a short overview over the components. The central element is the language model which is an implementation of a restricted Boltzmann machine. As input this machine can take a data object which will define the size and type of its input layer. This data object could be a file containing n-gram indices or the output of another Boltzmann machine for stacking deep models. The factored data class just uses multiple data objects and combines them into one data object making it easy to train factored models. Each file uses some abstract vocabulary to determine the indices each sample should turn on. These vocabularies can be word indices or as newly proposed in this work, the indices of the letter bigram contained in a word. If we now wanted to extend the framework we could for example just add a new vocabulary without changing the rest of the system. The CD-Trainer is able to train a language model using the Contrastive Divergence algorithm used in section 3.2.2 supporting different learning rate, momentum settings and the ability to stack models. There is a user interface which is used for connections from outside, for example the STTK decoder or reranking script will used this class to get the free energy of data or sentences.

[1]http://www.clean-code-developer.de/

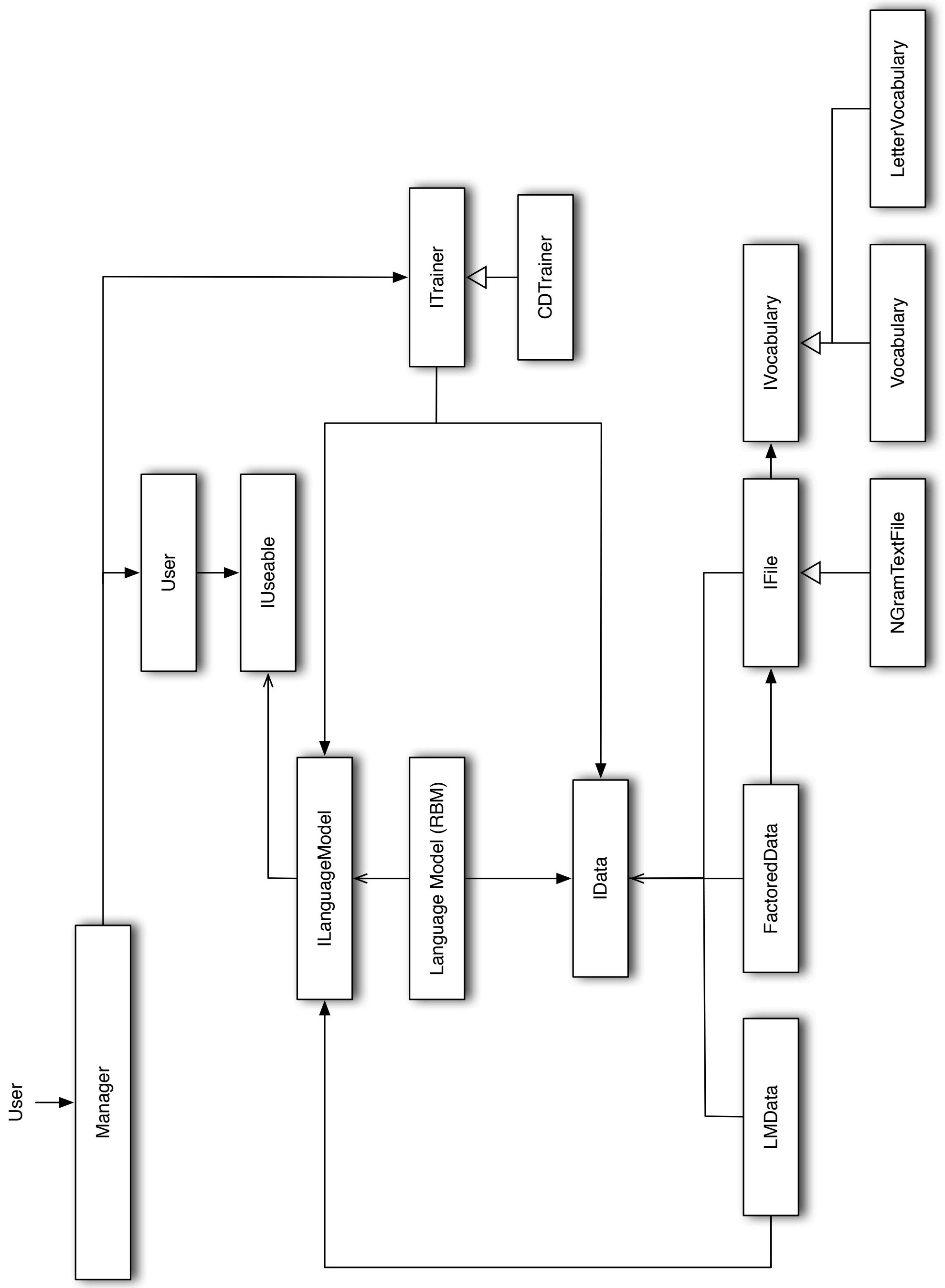

Figure 7.1.: A Sketch Design of the RBMLM.

7.2. Visual Debugging

During the implementation of this work we used different techniques for debugging the restricted Boltzmann machine network. Next to usual debugging tools like gdb (GNU debugger) and log statements that state the control flow we used techniques described in Yosinski and Lipson [YL12] to create a graphical debugging output of the network. In this Section we want to elaborate on these methods in detail. The first and for this work most important histogram was the plot of the activation of the hidden units, given different data samples. For this first a data sample is propagated into the hidden units and the hidden activation probability is then saved as a column of gray-scale pixels, where white means that the neuron is very likely to turn on and black means it is very unlikely. For each sample the hidden values are saved an then plotted into a picture. In the initialization of the model where all weights were chosen as random values from a zero-mean Gaussian with a deviation of about 0.01 as recommended in Hinton [Hin10]. In Figure 7.2 the hidden activations are shown. Each column is the activation of the 32 hidden units given the data sample with index of the row. The image is mostly gray, which means that the probability to activate for each feature detector is roughly 0.5 with some random noise. If you can see any patterns in this Image there might be a problem with the random initialization and also lots of white or black pixels mean that, probably the weights are initialized too large and the feature detectors already decided what features they are looking for without even seeing the data.

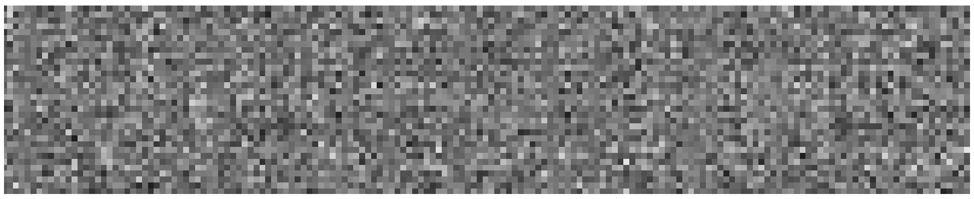

Figure 7.2.: Hidden activation histogram after initialization. In Y direction each pixel in each row corresponds to one hidden unit and in X direction each column corresponds to the hidden activation of one data sample.

After the first iteration of training over the whole data there should be visible changes in the hidden activations. This means there should be a tendency for some neurons to turn on and for others to turn off and some will remain undecided and remain gray. (See Figure 7.3)

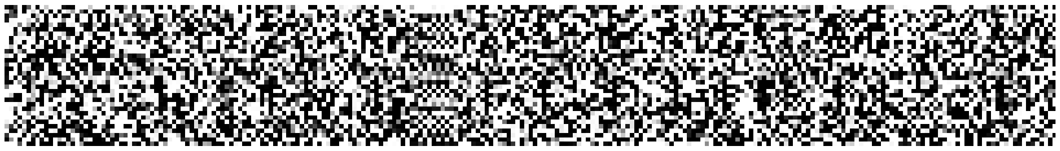

Figure 7.3.: Hidden activation histogram after the first iteration of training. In Y direction each pixel in each row corresponds to one hidden unit and in X direction each column corresponds to the hidden activation of one data sample.

If there are any patterns emerging it is possible that there is a bug in the implementation. The picture should mostly look like random noise and contain no visible forms. One reason can be that the training data was not shuffled before training or not shuffled for the histogram generation. Then depending on the task, in this example language modeling, some patterns can emerge at the boundaries between sentences. But with shuffled data this histogram should mostly be random noise.

In Figure 7.4 there are lots of vertical visible patterns and also most of the neurons are turned on. In this graphic maybe the weights were initialized too big or this output might be normal for the task. For this it might be important to take a look at the second or third iteration to see how the activations change.

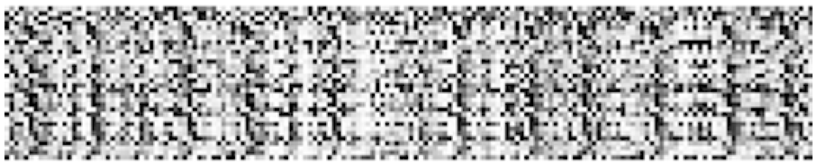

Figure 7.4.: Vertical Patterns are visible.

In Figure 7.5 are some feature detectors that are highly likely to turn on for every data sample. This is a bad because these features will not help us classify anything since they are always on anyway. This might point to a bug in the implementation and learning.

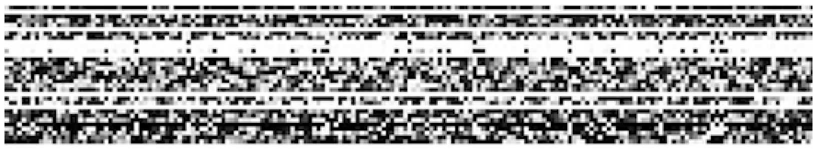

Figure 7.5.: Some units are on for all data samples.

Figure 7.6 is also an indicator of something not working right. We have a few data samples for which we get a repeating pattern and mostly gray in between. The pattern could be explained by sentence endings but the gray area in between would mean that our model can classify some features at sentence ends but is totally indecisive elsewhere, which is also unlikely. Since our data was shuffled, and the vertical visible patterns contain more than one sample it is highly unlikely that the sentence ending n-grams stick together like this. Maybe the random shuffling did not work correctly. Both the gray area and the patter that is repeating over multiple samples more than one time are strong indicators that there is a bug somewhere.

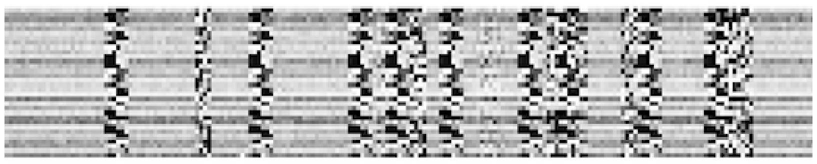

Figure 7.6.: The patterns that can be seen here look like there might be a bug. Depending on the data patterns like this can happen but should not. Every column represents a sample and the order of the samples were randomized so getting equal looking patterns should not happen that often. Also the gray pattern in between should not happen.

The last histogram we took a look at was the distribution of the visible and hidden biases and weights. Normally the visible biases get initialized from the data. In our case the mean value of the visible biases was 5.9. The hidden biases were initialized to zero and the weights as explained above randomly from a Gaussian distribution with mean value 0 and standard deviation of 0.1.

8. Evaluation

We evaluated the RBM-based language model on different statistical machine translation (SMT) tasks. We will first analyze what letter n-gram context is needed to be able to distinguish different words from the vocabulary well enough. Then we will give a brief description of our SMT system and afterwards, we will describe our experiments on the German-to-English TED and university lecture translation task in detail. After that we will describe the results of the English-to-German news translation system and the English-to-French TED task. We will conclude this chapter with a brief discussion about the size of the newly proposed models and their training time. Since we did most of the experiments on the German-to-English TED task a complete table showing all results for each of the three systems can be found in the Appendix A.

8.1. Word Representation

In first experiments we analyzed whether the letter-based representation is able to distinguish between different words. In a vocabulary of 27,748 words, we compared for different letter n-gram sizes how many words are mapped to the same input feature vector.

				#Vectors mapping to			
Model	Caps	VocSize	#Vectors	1 Word	2 Words	3 Words	4+ Words
WordIndex	-	27,748	27,748	27,748	0	0	0
Letter 1-gram	No	107	21,216	17,319	2,559	732	606
Letter 2-gram	No	1,879	27,671	27,620	33	10	8
Letter 3-gram	No	12,139	27,720	27,701	10	9	0
Letter 3-gram	Yes	8,675	27,710	27,681	20	9	0
Letter 4-gram	No	43,903	27,737	27,727	9	1	0
Letter 4-gram	Yes	25,942	27,728	27,709	18	1	0

Table 8.1.: Comparison of the vocabulary size and the possibility to have a unique representation of each word in the training corpus.

Table 8.1 shows the different models, their input dimensions and the total number of unique clusters as well as the amount of input vectors containing one, two, three or four or more words that get mapped to this input vector. In the word index representation every word has its own feature vector. In this case the dimension of the input vector is 27,748 and each word has its own unique input vector.

If we use only letters, as done in the unigram model, only 62% of the words have a unique representation. Furthermore, there are 606 feature vectors representing 4 or more words. This type of encoding of the words is not sufficient for the task.

When using a bigram letter context nearly each of the 27,748 words has a unique input representation, although the input dimension is only 7% compared to the word index. With the three letter vocabulary context and higher there is no input vector that represents more than three words from the vocabulary. This is good since we want similar words to be close together but not have exactly the same input vector. The words that are still clustered to the same input are mostly numbers or in case of the capital letter encoding typing mistakes like "YouTube" and "Youtube".

8.2. Translation System Description

The translation system for the German-to-English task was trained on the European Parliament corpus, News Commentary corpus, the BTEC corpus and TED talks[1]. The data was preprocessed and compound splitting was applied for German. Afterwards the discriminative word alignment approach as described in Niehues and Vogel [NV08] was applied to generate the alignments between source and target words. The phrase table was built using the scripts from the Moses package described in Koehn et al. [KHB+07]. A 4-gram language model was trained on the target side of the parallel data using the SRILM toolkit from Stolcke [Sto02]. In addition, we used a bilingual language model as described in Niehues et al. [NHVW11]. Reordering was performed as a preprocessing step using part-of-speech (POS) information generated by the TreeTagger [Sch94b]. We used the reordering approach described in Rottmann and Vogel [RV07] and the extensions presented in Niehues et al. [NK09] to cover long-range reorderings, which are typical when translating between German and English. An in-house phrase-based decoder was used to generate the translation hypotheses and the optimization was performed using the MERT implementation as presented in Venugopal et al. [VZW05]. All our evaluation scores are measured using the BLEU metric.

We trained the RBMLM models on 50K sentences from TED talks and optimized the weights of the log-linear model on a separate set of TED talks. For all experiments the RBMLMs have been trained with a context of four words. The development set consists of 1.7K segments containing 16K words. We used two different test sets to evaluate our models. The first test set contains TED talks with 3.5K segments containing 31K words. The second task was from an in-house computer science lecture (CSL) corpus containing 2.1K segments and 47K words. For both tasks we used the weights optimized on the TED data.

For the task of translating English news texts into German we used a system developed for the Workshop on Machine Translation (WMT) evaluation. The continuous space language models were trained on a random subsample of 100K sentences from the monolingual training data used for this task. The out-of-vocabulary rates for the TED task are 1.06% while the computer science lectures have 2.73% and nearly 1% on WMT.

[1]http://www.ted.com

8.3. German-to-English TED Task

The results for the translation of German TED lectures into English are shown in Table 8.2. The baseline system uses a 4-gram Kneser-Ney smoothed language model trained on the target side parallel data. We then added a RBMLM, which was only trained on the English side of the TED corpus.

If the word index RBMLM trained for one iteration using 32 hidden units is added, an improvement of about 1 BLEU point can be achieved. The letter bigram model performs about 0.3 BLEU points better than no additional model, but significantly worse than the word index model or the other letter n-gram models. The letter 3- to 5-gram-based models obtain similar BLEU scores, varying only by 0.1 BLEU point. They also achieve a 0.8 to 0.9 BLEU points improvement against the baseline system and a 0.2 to 0.1 BLEU points decrease than the word index-based encoding. The letter 6-gram model decreases in performance compared to the lower letter context models, even though the score on the development set is the highest of the letter n-gram models.

System	Dev	Test
Baseline	26.31	23.02
+WordIndex	**27.27**	**24.04**
+Letter 2-gram	26.66	23.30
+Letter 3-gram	26.80	23.84
+Letter 4-gram	26.79	23.93
+Letter 5-gram	26.64	23.82
+Letter 6-gram	26.85	23.66

Table 8.2.: Results for German-to-English TED translation task.

Using the word index model with the first baseline system increases the BLEU score nearly as much as adding a n-gram-based language model trained on the TED corpus as done in the baseline of the systems presented in Table 8.3. In these experiments all letter-based models outperformed the baseline system. The bigram-based language model performs worst and the 3- and 4-gram-based models perform only slightly worse than the word index-based model.

System	Dev	Test
Baseline+ngram	27.45	24.06
+WordIndex	**27.70**	**24.34**
+Letter 2-gram	27.45	24.15
+Letter 3-gram	27.52	24.25
+Letter 4-gram	27.60	24.30

Table 8.3.: Results of German-to-English TED translations using an additional in-domain language model.

A third experiment is presented in Table 8.4. In the baseline of this system we also applied phrase table adaptation as described in [NMH$^+$10]. In this experiment the word index model improves the system by 0.4 BLEU points. In this case all letter-based models perform very similar. They are again performing slightly worse than the word index-based system, but better than the baseline system.

System	Dev	Test
Baseline+ngram+adaptpt	28.40	24.57
+WordIndex	**28.55**	**24.96**
+Letter 2-gram	28.31	24.80
+Letter 3-gram	28.31	24.71
+Letter 4-gram	28.46	24.65

Table 8.4.: Results of German-to-English TED translations with additional in-domain language model and adapted phrase table.

To summarize the results, we could always improve the performance of the baseline system by adding the letter n-gram-based language model. Furthermore, in most cases, the bigram model performs worse than the higher order models. It seems to be important for this task to have more context information. The 3- and 4-gram-based models perform almost equal, but slightly worse than the word index-based model.

8.3.1. Caps Feature

In this subsection we evaluate the proposed caps feature compared to the non-caps letter n-gram model and the baseline systems. For the sake of clarity the tables this time contain the results of one letter n-gram context on all three system configurations.

In Table 8.5 the scores for the baselines, the letter bigram models and the caps features can be seen. It is notable that the capital letter variant improves the normal version except for the last system. On the first system the letter bigram using caps features is still around 0.3 BLEU points worse than the higher context letter n-gram models.

System	Dev	Test
Baseline	26.31	23.02
+Letter 2-gram	26.66	23.30
+Letter 2-gram caps	**26.67**	**23.44**
Baseline+ngram	27.45	24.06
+Letter 2-gram	27.45	24.15
+Letter 2-gram caps	**27.46**	**24.28**
Baseline+ngram+adaptpt	**28.40**	24.57
+Letter 2-gram	28.31	**24.80**
+Letter 2-gram caps	28.33	24.72

Table 8.5.: Difference between caps and non-caps letter n-gram models.

As we can see in Table 8.6 the caps features with the letter 3-gram model improve the baseline BLEU score by about ±0.2 BLEU points. As we saw in the case of the letter bigrams the 3-gram caps feature models perform slightly better than the normal version with the exception for the last task.

System	Dev	Test
Baseline	26.31	23.02
+Letter 3-gram	**26.80**	23.84
+Letter 3-gram caps	26.67	**23.85**
Baseline+ngram	27.45	24.06
+Letter 3-gram	27.52	24.25
+Letter 3-gram caps	**27.60**	**24.47**
Baseline+ngram+adaptpt	28.40	24.57
+Letter 3-gram	28.31	**24.71**
+Letter 3-gram caps	**28.43**	24.66

Table 8.6.: Difference between caps and non-caps letter n-gram models.

In Table 8.7 we can see the results for the letter 4-gram models. In this case the caps features only improve the last two systems. In the second system with the additional in-domain language model the letter 4-gram caps model improves the baseline by about 0.5 BLEU points which is even more than the word index model.

System	Dev	Test
Baseline	26.31	23.02
+Letter 4-gram	**26.79**	**23.93**
+Letter 4-gram caps	26.73	23.77
Baseline+ngram	27.45	24.06
+Letter 4-gram	**27.60**	24.30
+Letter 4-gram caps	**27.60**	**24.57**
Baseline+ngram+adaptpt	28.40	24.57
+Letter 4-gram	**28.46**	24.65
+Letter 4-gram caps	28.43	**24.73**

Table 8.7.: Difference between caps and non-caps letter n-gram models.

Concluding this subsection the caps features tend to improve the non-caps layers on the TED task. The improvement is on average around 0.08 BLEU points while the best system improves by 0.27 BLEU points. Even though the English language does not contain many capital letter words the capital letter manages to improve the input layer while also decreasing the input size and thus reducing the training time.

8.3.2. Tree-Split Feature

In this subsection we are going to evaluate the tree split features. We concluded experiments with a tree depth of one and two using both capital letter features and just letter n-gram features and different letter n-gram sizes. The results for the first baseline system can be seen in Table 8.8. We can see that the tree split features decrease the performance of the regular letter n-gram system on average by 0.2 BLEU points but still improve the baseline without RBMLM by about 0.65 BLEU points.

System	Dev	Test
Baseline	26.31	23.02
+Letter 3-gram	**26.80**	**23.84**
+Letter 3-gram 1-TreeDepth	26.47	23.65
+Letter 3-gram 2-TreeDepth	26.71	23.72
+Letter 4-gram	**26.79**	**23.93**
+Letter 4-gram 1-TreeDepth	**26.79**	23.67
+Letter 4-gram 2-TreeDepth	26.74	23.73

Table 8.8.: The results of the tree split features for the TED baseline system.

As we can see in Table 8.9 the results for the system with the additional in-domain language model show that depending on the system the tree split features can give an improvement over the conventional letter n-gram features. The performance of the tree split feature varies from minus 0.03 BLEU points to plus 0.1 BLEU points compared to the letter n-gram input layer baseline. Still all tree split systems increase the BLEU score of the baseline without continuous space language model.

System	Dev	Test
Baseline+ngram	27.45	24.06
+Letter 3-gram	**27.52**	24.25
+Letter 3-gram 1-TreeDepth	27.49	**24.32**
+Letter 3-gram 2-TreeDepth	27.38	24.22
+Letter 4-gram	27.60	24.30
+Letter 4-gram 1-TreeDepth	**27.66**	24.27
+Letter 4-gram 2-TreeDepth	27.63	**24.33**

Table 8.9.: The results of the tree split features for the TED system with additional in-domain language model.

In Table 8.10 for the third baseline system we can see that the tree split feature improves the letter 3-gram model. The best system, which is the letter 3-gram with tree depth of two improves over the normal input layer by 0.27 BLEU points, the same tree depth for the letter 4-gram decreases the score by 0.2 which is even below the normal system baseline.

System	Dev	Test
Baseline+ngram+adaptpt	28.40	24.57
+Letter 3-gram	28.31	24.71
+Letter 3-gram 1-TreeDepth	**28.33**	24.60
+Letter 3-gram 2-TreeDepth	**28.33**	**24.98**
+Letter 4-gram	28.46	**24.65**
+Letter 4-gram 1-TreeDepth	**28.48**	**24.65**
+Letter 4-gram 2-TreeDepth	28.38	24.47

Table 8.10.: The results of the tree split features for the TED system with additional in-domain language model and adapted phrase table.

After seeing the first experiments on the three baseline systems we conclude that except for the second TED system with the additional in-domain language model, the tree split feature does not work as well as the normal letter n-grams. This changes when we combine the tree split and capital letter feature as can be seen in Table 8.11. We can see that using the tree depth one as additional feature increases the BLEU score by about 0.05 BLEU points while using a tree depth of two performs around 0.2 BLEU points worse.

System	Dev	Test
Baseline	26.31	23.02
+Letter 3-gram caps	26.67	23.85
+Letter 3-gram caps 1-TreeDepth	26.66	**23.89**
+Letter 3-gram caps 2-TreeDepth	**26.69**	23.72
+Letter 4-gram caps	26.73	23.77
+Letter 4-gram caps 1-TreeDepth	**26.79**	**23.81**
+Letter 4-gram caps 2-TreeDepth	26.60	23.54

Table 8.11.: The results of the tree split and caps features for the TED system.

The results for the system with the additional in-domain language model can be seen in Table 8.12. In this system configuration the tree split models all perform worse than the newly proposed input layer without the tree split feature. The decrease for the letter 3-gram is about 0.07 BLEU points while the BLEU score of the letter 4-gram model decreases by about 0.25 BLEU points.

System	Dev	Test
Baseline+ngram	27.45	24.06
+Letter 3-gram caps	**27.60**	**24.47**
+Letter 3-gram caps 1-TreeDepth	27.50	24.40
+Letter 3-gram caps 2-TreeDepth	27.54	24.40
+Letter 4-gram caps	27.46	**24.57**
+Letter 4-gram caps 1-TreeDepth	**27.60**	24.37
+Letter 4-gram caps 2-TreeDepth	27.50	24.23

Table 8.12.: The results of the tree split and caps features for the TED system with additional in-domain language model.

The results for last configuration for the German-to-English TED task with additional in-domain language model and additional phrase table can be seen in Table 8.13. Most of the configurations are able to improve over the normal letter n-gram model. The average increase is about 0.05 BLEU points with the biggest increase being 0.12 BLEU points.

System	Dev	Test
Baseline+ngram+adaptpt	28.40	24.57
+Letter 3-gram caps	**28.43**	24.66
+Letter 3-gram caps 1-TreeDepth	28.37	24.61
+Letter 3-gram caps 2-TreeDepth	28.26	**24.77**
+Letter 4-gram caps	28.38	24.73
+Letter 4-gram caps 1-TreeDepth	28.35	**24.85**
+Letter 4-gram caps 2-TreeDepth	**28.39**	24.75

Table 8.13.: The results of the tree split and caps features for the TED system with additional in-domain language model and adapted phrase table.

Concluding this section, we found that all tree split except for one configuration improve the score of the baseline. In the sorted tables in Appendix A we can see that except for the first TED baseline system tree split systems on average perform better than the normal letter n-gram input layer and that combining capital letter features combined with tree split feature performs better on average than just using tree split features alone.

8.3.3. Iterations

As showed in Niehues and Waibel [NW12] training the RBMLM for more than one iteration might be useful. We trained one of our letter n-gram models for 10 iterations and took the 10th iteration for translation. In Table 8.14 we show the free energy for all the iterations. We can see that the free energy decreases between all iterations. After the first iteration the average free energy of the training set is -9.27141 while after the 10th iteration it decreased to -41.2808. On a held out development set the energy went down from -15.6257 to -48.1398 being lowest in the 10th iteration.

	Free Energy	
Iteration	Dev	Train
1	-15.6257	-9.27141
2	-30.4127	-23.6874
3	-32.7115	-25.8486
4	-30.5272	-23.7507
5	-45.7468	-38.7981
6	-45.5386	-39.3284
7	-46.8414	-38.4325
8	-48.0029	-40.0611
9	-48.1398	-41.0315
10	-48.1398	-41.2808

Table 8.14.: Changes of free energy during multiple iterations of training.

A low free energy in a restricted Boltzmann machine means a high probability, so the 10th iteration should be the best trained model for our task. The results can be seen in Table 8.15, where it performed as well as trained for one Iteration. There was an increase in performance for the word-index model, we could not find the letter n-gram model to improve on the TED task by training multiple iterations.

System	Dev	Test
Baseline	26.31	23.02
+WordIndex 1 Iter	27.27	24.04
+WordIndex 10 Iter	**27.61**	**24.47**
+Letter 3-gram 1 Iter	26.80	23.84
+Letter 3-gram 10 Iter	26.76	23.85

Table 8.15.: The results for multiple iteration training on German-to-English TED task.

Concluding this section, we could not find the same improvement, as for the word index model, training for multiple iterations using our newly proposed input layer. Since the free energy of the training and development set decreased during training we can conclude that the model trained for more iterations models the data better than the model trained for only one Iteration. Not seeing an increase in BLEU score can have different reasons and more experiments of different Systems would have to be done.

8.3.4. Hidden Size

In this subsection we want to present the results of the models having more than 32 hidden units. We concluded the experiment on the first of the three German-to-English TED systems and the results can be seen in Table 8.16. We chose to compare the 32 hidden unit model with 64 and 128 hidden units.

System	Dev	Test
Baseline	26.31	23.02
+WordIndex 32H	**27.27**	**24.04**
+Letter 3-gram 32H	26.80	23.84
+Letter 3-gram 64H	26.95	23.82
+Letter 3-gram 128H	26.83	23.85

Table 8.16.: Results for German-to-English TED translation task.

The results show that increasing the hidden size did not increase the BLEU score on the test set, while with 64 hidden units a small gain could be seen on the development data.

8.3.5. Combination of Models

Another experiment concerning the combination of models was done. We wanted to see if the combination of the word index model, which performs well as a general language model and the letter n-gram model, which tries improve on out-of-vocabulary words and morphological word forms, could achieve better results.

For this we tried using both of the models as a log-linear feature in the decoder and also trained a new model containing both input layers. The results of this can be seen in Table 8.17. The jointly trained models decrease the performance of the separate models slightly. It maybe that the joint training of the models could benefit from an additional training iteration or a larger hidden size. The log linear combinations of both models on the other hand improved over the word index by 0.11 BLEU points. Concluding this subsection we found that the combination of models can improve the gain in BLEU score even further than using each model on its own.

System	Dev	Test
Baseline	26.31	23.02
+WordIndex	27.27	24.04
+Letter 3-gram	26.80	23.84
+WordIndex+Letter 3-gram	**27.28**	**24.11**
+WordIndex+Letter 3-gram caps	27.26	24.00
+WordIndex+Letter 3-gram joint	26.61	23.76
+WordIndex+Letter 3-gram caps joint	26.68	23.79

Table 8.17.: Results of combined models as different features in the decoder. These experiments were performed on the German-to-English TED task.

8.3.6. Deep Belief Models

We created several deep neural language models and used different input layers and hidden layer configurations. The results of the deep models used on the most basic TED baseline can be seen in Table 8.18.

System	Dev	Test
Baseline	26.31	23.02
+WordIndex	**27.27**	**24.04**
+WordIndex.8H.8H.8H.32H	26.73	23.86
+WordIndex.16H.8H.16H.32H	26.89	23.62
+Letter 3-gram	**26.80**	23.84
+Letter 3-gram.8H.8H.8H.32H	26.64	**23.88**
+Letter 3-gram.16H.8H.16H.32H	26.60	23.64

Table 8.18.: Deep belief model results on German-to-English TED task.

The deep models with the bottleneck architecture for the propagation of the word history decreased by about 0.2-0.4 BLEU points compared to the baseline continuous space model, they still improved the system baseline scores by about 0.6 BLEU points. The layout propagating the word history through multiple layers of 8 hidden units decreased the score of the word index model by about 0.2 BLEU points while the deep model with the letter 3-gram input stayed the same. In general what we can conclude from this is that there need to be more experiments with different layouts for the deep model. Maybe the model with multiple layers of 8 hidden units can be further improve by increasing the size of the hidden layers.

8.3.7. Unknown Words

We will further research if the newly proposed layer can improve the use of unknown words in the translations. For this we counted the words that were used in the translations, but not inside the training vocabulary of the German-to-English TED task. Of those words we counted how many were actually also inside the reference and compared the baseline and word index with our input layers. The Results for our first TED configurations can be seen in Table 8.19. In the column #UnkHit we can see the number of unknown words, that were also inside the reference. In the column #UnkMiss we can see the number of unknown words in the translation hypothesis that were not inside the reference translation. We can see that the system with the highest BLEU score has the lowest number of correctly translated unknown words. We can also note that it is the system with the lowest erroneous translated unknown words. From this table it seems that the increase in BLEU score depends more on the decrease of wrong translated unknowns than the correct translation of unknown words.

System	#UnkHit	#UniqeHit	#UnkMiss	#UniqueMiss	BLEU
Baseline	184	121	468	379	23.02
WordIndex	177	115	420	345	**24.04**
Letter 3-gram caps	183	119	457	365	23.85
Letter 4-gram caps	182	118	452	367	23.77

Table 8.19.: Unknown words in hypothesis of the German-to-English TED system. The table shows both, correct use of an unknown word and the use of unknown words that were not inside the reference.

In the second configuration seen in Table 8.20 we can see that the unknowns that are inside the translations and the reference is again highest for the baseline. The wrong unknown words are again lowest for the word index model being nearly 200 less than the baseline. In this task even though the BLEU score increased the number of mistaken unknown words also increased overall and is lowest for the word-index model.

System	#UnkHit	#UniqeHit	#UnkMiss	#UniqueMiss	BLEU
Baseline+ngram	189	126	643	529	24.06
WordIndex	180	118	446	368	24.34
Letter 3-gram caps	186	123	539	439	24.47
Letter 4-gram caps	183	120	508	421	**24.57**

Table 8.20.: Unknown words in hypothesis of the German-to-English TED system with additional in-domain language model. The table shows both, correct use of an unknown word and mistranslations.

In Table 8.21 we can see the results for the configuration with additional language model and adapted phrase table. The number of unknown words used in the translation and reference is almost equal for all systems, while the number of wrong unknowns used is lowest for the letter 4-gram model using caps features.

System	#UnkHit	#UniqeHit	#UnkMiss	#UniqueMiss	BLEU
Baseline+ngram+adaptpt	180	118	435	370	24.57
WordIndex	180	118	428	373	**24.96**
Letter 3-gram caps	179	117	419	357	24.66
Letter 4-gram caps	180	118	414	353	24.73

Table 8.21.: Unknown words in hypothesis of the German-to-English TED system with additional in-domain language model and adapted phrase table. The table shows both, correct use of an unknown word and mistranslations.

In Table 8.22 the total number of unknown words and words in the reference is listed. From a total of 3719 unique words in the reference we have 372 (10 %) unknown words that are not inside the vocabulary of the RBMLM. Out of these 372 words about 120 (32%) were translated correctly in every configuration.

	Reference
Total Unk	506
Unique Unk	372
Total Words	32184
Unique Words	3719

Table 8.22.: Number of Tokens not inside vocabulary

Concluding this section we can say that on TED there was no obvious connection between the number of correct and incorrect used unknown words. In the first configurations the system with the best BLEU score had the least incorrect used unknown words, while in the second configuration the best system had neither the most correct used nor the least incorrect used unknown words. On average the letter n-gram layer more correct unknown words than the word index input layer, but also uses more incorrect unknown words.

8.4. German-to-English CSL Task

After that, we evaluated the computer science lecture (CSL) test set. We used the same system as for the TED translation task. We did not perform a new optimization, since we wanted so see how well the models performed on a different task.

The results are summarized in Table 8.23. In this case the baseline is outperformed by the word index approach by approximately 1.1 BLEU points. Except for the 4-gram model the results are similar to the result for the TED task. All systems could again outperform the baseline.

System	Test
Baseline	23.60
+WordIndex	**24.76**
+Letter 2-gram	24.17
+Letter 3-gram	24.36
+Letter 4-gram	23.82

Table 8.23.: Results the baseline of the German-to-English CSL task.

The system with the additional in-domain language model in Table 8.24 shows that both letter n-gram language models perform better than the baseline and the word index model, improving the baseline by about 0.8 to 1 BLEU points. Whereas the word index model only achieved an improvement of 0.6 BLEU points.

System	Test
Baseline+ngram	23.81
+WordIndex	24.41
+Letter 2-gram	24.37
+Letter 3-gram	24.66
+Letter 4-gram	**24.85**

Table 8.24.: Results on German-to-English CSL corpus with additional in-domain language model.

The results of the system with the additional phrase table adaption can be seen in Table 8.25. The word index model improves the baseline by 0.25 BLEU points. The letter n-gram models improve the baseline by about 0.3 to 0.4 BLEU points also improving over the word index model. The letter bigram model in this case performs worse than the baseline.

System	Test
BL+ngram+adaptpt	25.00
+WordIndex	25.25
+Letter 2-gram	24.68
+Letter 3-gram	**25.43**
+Letter 4-gram	25.33

Table 8.25.: Results on German-to-English CSL with additional in-domain language model and adapted phrase table.

In summary, again the 3- and 4-gram letter models perform mostly better than the bigram version. They both almost equal. In contrast to the TED task, they were even able to outperform the word index model in some configurations by up to 0.4 BLEU points.

8.5. English-to-German News Task

The English-to-German news task was released for the Workshop on Statistical Machine Translation (WMT) 2013 conference. It consists of English-to-German news translations. We expect our input encoding to work better on this task since German has more word morphologies than the English language. The system presented in Table 8.26 uses discriminative word alignment, a bilingual language model, and a class-based language model as well as a discriminative word lexicon. With this system we could not improve the performance of the baseline by using a word index model, but in contrast the performance dropped by 0.1 BLEU points. If we use a letter bigram model, we could improve the translation quality by 0.1 BLEU points over the baseline system.

System	Dev	Test
Baseline	16.90	17.36
+WordIndex	16.79	17.29
+Letter 2-gram	**16.91**	**17.48**

Table 8.26.: Results for WMT2013 task English-to-German news.

The system presented in Table 8.27 has an additional POS-based language model and uses lexicalized reordering rules as well as tree rules for reordering. The letter bigram model is able to improve over the baseline by about 0.14 BLEU points.

System	Dev	Test
Baseline	**17.21**	17.59
+Letter 2-gram	16.91	**17.73**

Table 8.27.: Results for WMT2013 task English-to-German news.

Concluding this section we were able to improve the baseline on a English-to-German task. In this task, similar to the CSL lectures the word-index RBMLM failed to improve the baseline while the letter n-gram model showed to give an increase of about 0.1 BLEU points. Further experiments with higher letter n-gram context have to be concluded to see if the BLEU score can be increased even further.

8.6. English-to-French TED Task

In this section we are going to evaluate the English-to-French translation task. In Table 8.28 we can see that the RBMLMs, except for the letter 2-gram model, provide only a very small increase in BLEU score. The best model is the letter 3-gram with capital letter feature increasing BLEU by about 0.07 points.

System	Dev	Test
Baseline	29.15	32.13
+WordIndex	29.17	32.18
+Letter 2-gram	29.17	31.92
+Letter 2-gram caps	29.21	31.89
+Letter 3-gram	**29.25**	31.92
+Letter 3-gram caps	28.98	**32.20**

Table 8.28.: Results for English-to-French TED translation task

Concluding this section we found that the French baseline could only be improved by the WordIndex and letter 3-gram model using caps features. It is interesting to note that even though the letter 3-gram caps model performed worse on the development data than any other system, it achieved the best BLEU score on the test data.

8.7. Model Size and Training Times

In general the letter n-gram models perform almost as good as the word index model on English language tasks. The advantage of the models up to the letter 3-gram context model is that the training time is lower compared to the word index model. All the models were trained using 10 cores and a batch size of 10 samples per Contrastive Divergence update. As can be seen in Table 8.29 the letter 3-gram model needs less than 50% of the weights and takes around 75% of the training time of the word index model. The four letter n-gram model takes longer to train due to more parameters.

Even though the letter 5-gram model has 12.5 million parameters the performance was not as good as the lower context models. With this many parameters it is harder to find good values for the weights, which means that more training could prove to improve the performance. We can also see that the letter 2-gram model is very small and easy to train. This might be good for big systems like we saw in the English-to-German WMT task where the letter 2-gram model even improved over the word index model. The deep belief networks took less than half of the time the corresponding basic model needed. This shows great potential since the performance of the deep letter 3-gram models was nearly as good as the normal letter 3-gram model.

Model	#Weights	Time
WordIndex	3.55 M	20 h 10 min
Letter 2-gram	0.24 M	1h 24 min
Letter 3-gram	1.55 M	15 h 12 min
Letter 3-gram caps	1.11 M	10 h 32 min
Letter 3-gram 1-TreeDepth	3.3 M	20 h 11 min
Letter 3-gram caps 1-TreeDepth	2.5 M	18 h 43 min
Letter 3-gram 2-TreeDepth	4.9 M	27 h 34 min
Letter 3-gram caps 2-TreeDepth	3.7 M	24 h 07 min
Letter 4-gram	5.62 M	38 h 59 min
Letter 5-gram	12.53 M	63 h 01 min
Letter 3-gram.8H.8H.8H.32H	0.68 M	6 h 18 min
Letter 3-gram.16H.8H.16H.32H	0.97 M	8 h 04 min

Table 8.29.: Training time and number of parameters of the RBMLM models.

9. Future Work & Conclusion

In this work we developed new input encoding features for continuous space language models. The so called letter n-gram approach uses sub-word letter n-grams to represent words and help to still evaluate words that were never seen in the vocabulary. We think that this input layer helps to learn different morphological features inside the hidden representation of the network. We also found that it is useful to add additional feature detectors that represent whether the word is written in capital letters or even all capital letters, e.g. abbreviations. The last feature we tried was the Tree-Split algorithm. We wanted to see if it is helpful to store more data about the position of the letter n-gram inside the word. For this we split the word in half creating a tree structure where in each depth the letter n-grams get additional information appended. We did several experiments to explore if the newly proposed input layers can achieve an improvement over common approach input layouts. We evaluated if the input layer is able to classify unknown words better and give more accurate probabilities for word n-grams. We compared them using multiple languages and different translation systems. For this we used lecture translation tasks using the TED dataset and the in-house computer science lecture corpus and on a system used for the WMT 2013 conference. The languages used were French, English and German. In every setup the normal letter n-gram approach was able to improve the baseline and depending on the task even outperform the word index model. If we use additional features like capital letter information or the tree split feature we were sometimes able to outperform the word index model and normal letter n-gram models. The capital letter feature reduced the parameters to train even more while generalizing over a bigger word space. On average the capital letter feature helped to improve the normal letter n-gram layer even on a language like English, where only few capitalized words are used. For the tree split feature we saved the order in which the letter n-grams appeared in a tree structure to see if the information that certain letter n-grams are more probable at the end of the word than in the beginning is useful. This feature on the contrary increased the input size of the model while in one case even decreasing the performance compared to the baseline. In general the performance of the tree split models were good while we believe that a tree depth of one and additional capital letter feature achieved the best performance of all tree split feature systems on average.

We believe the optimal n-gram context to be three for the English language and instead of increasing the letter n-gram context rather increase the hidden size and training iterations. Even though the experiments showed that on the German-to-English TED task training for 10 iterations did not increase the BLEU score, the free energy of test and development data decreased, which means that the average probability of those sentences increased.

Another advantage of the subword feature layer is, that the input layer size and hence the training time and number of parameters is reduced a decrease to 75 % of the training time can be achieved. In some cases a bigram letter model achieved better scores than the common word index approach using only a tenth of the training time.

The second part of this work researched a layout for a deep belief network to classify n-grams. In the initial experiments we found that the performance compared to the shallow architecture using the same input layer decreased on average. But this might be due to the fact that the deep models we trained had far less parameters than the shallow architectures. The problem with bigger networks was that the evaluation time of the n-gram probabilities while decoding took very long. We believe that more experiments using bigger models has to be done to make a qualified statement about the chosen architecture.

In the future more experiments of different systems and languages would have to be concluded to see if the models provide even more useful on languages like Arabic, where lots of word morphologies are used. We would also like to do more experiments using the deep layout and maybe even compare more deep layouts with each other. We also believe that putting more effort in joint training of different input layers might increase the gain in BLEU points. Porting the learning algorithm onto a GPU could help to make the training of even bigger models or training more iterations feasible. Using additional features like part-of-speech could help to further improve the letter n-gram models. The shallow RBM structure in general might be useful to see if translation models can be learned. This has been done with feed forward neural networks [SAY12, Sch12], but not with shallow restricted Boltzmann machine models.

Bibliography

[BDVJ03] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A Neural Probabilistic Language Model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[CG96] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," in *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, ser. ACL '96. Stroudsburg, PA, USA: Association for Computational Linguistics, 1996, pp. 310–318.

[CPH] M. A. Carreira-Perpinan and G. E. Hinton, "On Contrastive Divergence Learning."

[DYDA12] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition," *IEEE Transactions on Audio, Speech & Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.

[EZM08] A. Emami, I. Zitouni, and L. Mangu, "Rich morphology based n-gram language models for Arabic," in *INTERSPEECH.* ISCA, 2008, pp. 829–832.

[Gal91] S. I. Gallant, "A Practical Approach for Representing Context and for Performing Word Sense Disambiguation Using Neural Networks," *Neural Computation*, vol. 3, no. 3, pp. 293–309, 2013/03/14 1991.

[HDY$^+$12] G. Hinton, L. Deng, D. Yu, G. Dahl, A. rahman Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," *Signal Processing Magazine*, 2012.

[Hin02] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, Aug. 2002.

[Hin10] G. Hinton, "A Practical Guide to Training Restricted Boltzmann Machines," University of Toronto, Tech. Rep., 2010.

[HOT06] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, July 2006.

[HS06] G. Hinton and R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, no. 5786, pp. 504 – 507, 2006.

[Kat87] S. M. Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer," in *IEEE Transactions on Acoustics, Speech and Signal Processing*, 1987, pp. 400–401.

[KHB$^+$07] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst, "Moses: Open Source Toolkit for Statistical Machine Translation," in *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ser. ACL '07, Stroudsburg, PA, USA, 2007, pp. 177–180.

[KN95] R. Kneser and H. Ney, "Improved backing-off for n-gram language modeling," in *In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. I, Detroit, Michigan, May 1995, pp. 181–184.

[KY05] K. Kirchhoff and M. Yang, "Improved Language Modeling for Statistical Machine Translation," in *Proceedings of the ACL Workshop on Building and Using Parallel Texts*, Ann Arbor, Michigan, USA, 2005, pp. 125–128.

[LOM+11] H. S. Le, I. Oparin, A. Messaoudi, A. Allauzen, J.-L. Gauvain, and F. Yvon, "Large Vocabulary SOUL Neural Network Language Models," in *INTERSPEECH.* ISCA, 2011, pp. 1469–1472.

[MD91] R. Miikkulainen and M. G. Dyer, "Natural Language Processing with Modular PDP Networks and Distributed Lexicon," *Cognitive Science*, vol. 15, pp. 343–399, 1991.

[MH07] A. Mnih and G. Hinton, "Three new graphical models for statistical language modelling," in *Proceedings of the 24th International Conference on Machine Learning*, 2007, pp. 641–648.

[MKB+10] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, "Recurrent Neural Network Based Language Model," in *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan*, 2010, pp. 1045–1048.

[Mni10] A. Mnih, "Learning Distributed Representations For Statistical Language Modelling And Collaborative Filtering," Ph.D. dissertation, University of Toronto, Toronto, Ont., Canada, Canada, 2010.

[Nea92] R. M. Neal, "Connectionist learning of belief networks," *Artif. Intell.*, vol. 56, no. 1, pp. 71–113, Jul. 1992.

[NHVW11] J. Niehues, T. Herrmann, S. Vogel, and A. Waibel, "Wider context by using bilingual language models in machine translation," in *Proceedings of the Sixth Workshop on Statistical Machine Translation*, ser. WMT '11. Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 198–206.

[NK09] J. Niehues and M. Kolss, "A POS-based model for long-range reorderings in SMT," in *Proceedings of the Fourth Workshop on Statistical Machine Translation*, ser. StatMT '09. Stroudsburg, PA, USA: Association for Computational Linguistics, 2009, pp. 206–214.

[NMH+10] J. Niehues, M. Mediani, T. Herrmann, M. Heck, C. Herff, and A. Waibel, "The KIT Translation system for IWSLT 2010," in *Proceedings of the seventh International Workshop on Spoken Language Translation (IWSLT)*, Paris, France, 2010.

[NMKS90] M. Nakamura, K. Maruyama, T. Kawabata, and K. Shikano, "Neural network approach to word category prediction for English texts," in *Proceedings of the 13th conference on Computational linguistics - Volume 3*, ser. COLING '90, Stroudsburg, PA, USA, 1990, pp. 213–218.

[NV08] J. Niehues and S. Vogel, "Discriminative word alignment via alignment matrix modeling," in *Proceedings of the Third Workshop on Statistical Machine Translation*, ser. StatMT '08, Stroudsburg, PA, USA, 2008, pp. 18–25.

[NW12] J. Niehues and A. Waibel, “Continuous Space Language Models using Restricted Boltzmann Machines,” in *Proceedings of the International Workshop for Spoken Language Translation (IWSLT 2012)*, Hong Kong, 2012.

[rMDH12] A. rahman Mohamed, G. E. Dahl, and G. E. Hinton, “Acoustic Modeling Using Deep Belief Networks,” *IEEE Transactions on Audio, Speech & Language Processing*, vol. 20, no. 1, pp. 14–22, 2012.

[RV07] K. Rottmann and S. Vogel, “Word Reordering in Statistical Machine Translation with a POS-Based Distortion Model,” in *TMI*, Skövde, Sweden, 2007.

[SAY12] L. H. Son, A. Allauzen, and F. Yvon, “Continuous Space Translation Models With Neural Networks,” in *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, ser. NAACL HLT ’12. Stroudsburg, PA, USA: Association for Computational Linguistics, 2012, pp. 39–48.

[Sch94a] H. Schmid, “Part-Of-Speech Tagging With Neural Networks,” in *Proceedings of the 15th conference on Computational linguistics - Volume 1*, ser. COLING ’94. Stroudsburg, PA, USA: Association for Computational Linguistics, 1994, pp. 172–176.

[Sch94b] ——, “Probabilistic Part-of-Speech Tagging Using Decision Trees,” in *International Conference on New Methods in Language Processing*, Manchester, UK, 1994.

[Sch07] H. Schwenk, “Continuous Space Language Models,” *Comput. Speech Lang.*, vol. 21, no. 3, pp. 492–518, July 2007.

[Sch10] ——, “Continuous-Space Language Models for Statistical Machine Translation,” *Prague Bull. Math. Linguistics*, vol. 93, pp. 137–146, 2010.

[Sch12] ——, “Continuous Space Translation Models for Phrase-Based Statistical Machine Translation,” in *COLING (Posters)*, M. Kay and C. Boitet, Eds. Indian Institute of Technology Bombay, 2012, pp. 1071–1080.

[SD07] R. Sarikaya and Y. Deng, “Joint Morphological-Lexical Language Modeling for Machine Translation,” in *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, Rochester, New York, USA, 2007, pp. 145–148.

[SG05] H. Schwenk and J.-L. 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*, ser. HLT ’05. Stroudsburg, PA, USA: Association for Computational Linguistics, 2005, pp. 201–208.

[SMSN11] M. A. B. Shaik, A. E.-D. Mousa, R. Schlüter, and H. Ney, “Hybrid Language Models Using Mixed Types of Sub-Lexical Units for Open Vocabulary German lvcsr,” in *INTERSPEECH 2011, 12th Annual Conference of the International Speech Communication Association, Florence, Italy.*, 2011.

[SRA12] H. Schwenk, A. Rousseau, and M. Attik, “Large, Pruned or Continuous Space Language Models on a GPU for Statistical Machine Translation,” in *NAACL Workshop on the Future of Language Modeling*, June 2012.

[SSG10] H. Sak, M. Saraclar, and T. Guengoer, “Morphology-based and Sub-word Language Modeling for Turkish Speech Recognition,” in *2010 IEEE International*

Conference on Acoustics Speech and Signal Processing (ICASSP), 2010, pp. 5402–5405.

[Sto02] A. Stolcke, “SRILM - An Extensible Language Modeling Toolkit,” in *7th International Conference on Spoken Language Processing, ICSLP2002/INTERSPEECH 2002, Denver, Colorado, USA.*, 2002.

[VZW05] A. Venugopal, A. Zollmann, and A. Waibel, “Training and Evaluating Error Minimization Rules for Statistical Machine Translation,” in *Proceedings of the ACL Workshop on Building and Using Parallel Texts*, ser. ParaText ’05. Stroudsburg, PA, USA: Association for Computational Linguistics, 2005, pp. 208–215.

[WHH$^+$90] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, “Phoneme Recognition using Time-delay Neural Networks,” in *Readings in Speech Recognition*, A. Waibel and K.-F. Lee, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, pp. 393–404.

[XR00] W. Xu and A. Rudnicky, “Can Artificial Neural Networks learn Language Models?” in *Sixth International Conference on Spoken Language Processing, ICSLP 2000 / INTERSPEECH 2000, Beijing, China.* ISCA, 2000, pp. 202–205.

[YL12] J. Yosinski and H. Lipson, “Visually Debugging Restricted Boltzmann Machine Training with a 3D Example,” in *Presented at Representation Learning Workshop, 29th International Conference on Machine Learning*, 2012.

Appendix

A. Complete Tables for each Task

In this section we want to show the complete tables for the German-to-English TED task. The first two tables show the basic baseline and the experiments done. The next two tables show the experiments for the task with the additional in-domain language model and the last two tables show the experiments for the additional language model and adapted phrase table. The first of each Table is sorted by experiments with the best scores marked as bold and the second tables are sorted by ascending BLEU scores.

System	Dev	Test
Baseline	26.31	23.02
+WordIndex	27.27	24.04
+WordIndex 10 Iter	**27.61**	**24.47**
+WordIndex+Letter 3-gram	27.28	24.11
+WordIndex+Letter 3-gram caps	27.26	24.00
+WordIndex.8H.8H.8H.32H	26.73	23.86
+WordIndex.16H.8H.16H.32H	26.89	23.62
+Letter 2-gram	26.66	23.30
+Letter 2-gram caps	26.67	23.44
+Letter 3-gram	26.80	23.84
+Letter 3-gram 10 Iter	26.76	23.85
+Letter 3-gram.8H.8H.8H.32H	26.64	23.88
+Letter 3-gram.16H.8H.16H.32H	26.60	23.64
+Letter 3-gram caps	26.67	23.85
+Letter 3-gram 1-TreeDepth	26.47	23.65
+Letter 3-gram 2-TreeDepth	26.71	23.72
+Letter 3-gram caps 1-TreeDepth	26.66	23.89
+Letter 3-gram caps 2-TreeDepth	26.69	23.72
+Letter 4-gram	26.79	23.93
+Letter 4-gram caps	26.73	23.77
+Letter 4-gram 1-TreeDepth	26.79	23.67
+Letter 4-gram 2-TreeDepth	26.74	23.73
+Letter 4-gram caps 1-TreeDepth	26.79	23.81
+Letter 4-gram caps 2-TreeDepth	26.60	23.54
+Letter 5-gram	26.64	23.82

Table A.1.: Results for German-to-English TED translation task for all experiments, sorted by task.

System	Dev	Test
Baseline	26.31	23.02
+Letter 2-gram	26.66	23.30
+Letter 2-gram caps	26.67	23.44
+Letter 4-gram caps 2-TreeDepth	26.60	23.54
+WordIndex.16H.8H.16H.32H	26.89	23.62
+Letter 3-gram.16H.8H.16H.32H	26.60	23.64
+Letter 3-gram 1-TreeDepth	26.47	23.65
+Letter 4-gram 1-TreeDepth	26.79	23.67
+Letter 3-gram caps 2-TreeDepth	26.69	23.72
+Letter 3-gram 2-TreeDepth	26.71	23.72
+Letter 4-gram 2-TreeDepth	26.74	23.73
+Letter 4-gram caps	26.73	23.77
+Letter 4-gram caps 1-TreeDepth	26.79	23.81
+Letter 5-gram	26.64	23.82
+Letter 3-gram	26.80	23.84
+Letter 3-gram caps	26.67	23.85
+Letter 3-gram 10 Iter	26.76	23.85
+WordIndex.8H.8H.8H.32H	26.73	23.86
+Letter 3-gram.8H.8H.8H.32H	26.64	23.88
+Letter 3-gram caps 1-TreeDepth	26.66	23.89
+Letter 4-gram	26.79	23.93
+WordIndex+Letter 3-gram caps	27.26	24.00
+WordIndex	27.27	24.04
+WordIndex+Letter 3-gram	27.28	24.11
+WordIndex 10 Iter	**27.61**	**24.47**

Table A.2.: Results for German-to-English TED translation task for all experiments, sorted by ascending BLEU score .

System	Dev	Test
Baseline+ngram	27.45	24.06
+WordIndex	**27.70**	24.34
+Letter 2-gram	27.45	24.15
+Letter 2-gram caps	27.46	24.28
+Letter 3-gram	27.52	24.25
+Letter 3-gram caps	27.60	24.47
+Letter 3-gram 1-TreeDepth	27.49	24.32
+Letter 3-gram 2-TreeDepth	27.38	24.22
+Letter 3-gram caps 1-TreeDepth	27.50	24.40
+Letter 3-gram caps 2-TreeDepth	27.54	24.40
+Letter 4-gram	27.60	24.30
+Letter 4-gram caps	27.60	**24.57**
+Letter 4-gram 1-TreeDepth	27.66	24.27
+Letter 4-gram 2-TreeDepth	27.63	24.33
+Letter 4-gram caps 1-TreeDepth	27.60	24.37
+Letter 4-gram caps 2-TreeDepth	27.50	24.23

Table A.3.: Results for German-to-English TED translation task using an additional in-domain language model for all experiments, sorted by task.

System	Dev	Test
Baseline+ngram	27.45	24.06
+Letter 2-gram	27.45	24.15
+Letter 3-gram 2-TreeDepth	27.38	24.22
+Letter 4-gram caps 2-TreeDepth	27.50	24.23
+Letter 3-gram	27.52	24.25
+Letter 4-gram 1-TreeDepth	27.66	24.27
+Letter 2-gram caps	27.46	24.28
+Letter 4-gram	27.60	24.30
+Letter 3-gram 1-TreeDepth	27.49	24.32
+Letter 4-gram 2-TreeDepth	27.63	24.33
+WordIndex	**27.70**	24.34
+Letter 4-gram caps 1-TreeDepth	27.60	24.37
+Letter 3-gram caps 1-TreeDepth	27.50	24.40
+Letter 3-gram caps 2-TreeDepth	27.54	24.40
+Letter 3-gram caps	27.60	24.47
+Letter 4-gram caps	27.60	**24.57**

Table A.4.: Results for German-to-English TED translation task using an additional in-domain language model for all experiments, sorted by ascending BLEU scores.

System	Dev	Test
Baseline+ngram+adaptpt	28.40	24.57
+WordIndex	**28.55**	24.96
+Letter 2-gram	28.31	24.80
+Letter 2-gram caps	28.33	24.72
+Letter 3-gram	28.31	24.71
+Letter 3-gram caps	28.43	24.66
+Letter 3-gram 1-TreeDepth	28.33	24.60
+Letter 3-gram 2-TreeDepth	28.33	**24.98**
+Letter 3-gram caps 1-TreeDepth	28.37	24.61
+Letter 3-gram caps 2-TreeDepth	28.26	24.77
+Letter 4-gram	28.46	24.65
+Letter 4-gram caps	28.43	24.73
+Letter 4-gram 1-TreeDepth	28.48	24.65
+Letter 4-gram 2-TreeDepth	28.38	24.47
+Letter 4-gram caps 1-TreeDepth	28.35	24.85
+Letter 4-gram caps 2-TreeDepth	28.39	24.75

Table A.5.: Results for German-to-English TED translation task using an additional in-domain language model for all experiments, sorted by task.

System	Dev	Test
+Letter 4-gram 2-TreeDepth	28.38	24.47
Baseline+ngram+adaptpt	28.40	24.57
+Letter 3-gram 1-TreeDepth	28.33	24.60
+Letter 3-gram caps 1-TreeDepth	28.37	24.61
+Letter 4-gram	28.46	24.65
+Letter 4-gram 1-TreeDepth	28.48	24.65
+Letter 3-gram caps	28.43	24.66
+Letter 3-gram	28.31	24.71
+Letter 2-gram caps	28.33	24.72
+Letter 4-gram caps	28.43	24.73
+Letter 4-gram caps 2-TreeDepth	28.39	24.75
+Letter 3-gram caps 2-TreeDepth	28.26	24.77
+Letter 2-gram	28.31	24.80
+Letter 4-gram caps 1-TreeDepth	28.35	24.85
+WordIndex	**28.55**	24.96
+Letter 3-gram 2-TreeDepth	28.33	**24.98**

Table A.6.: Results for German-to-English TED translation task using an additional in-domain language model for all experiments, sorted by ascending BLEU scores.