Addressing the Out-of-Vocabulary Problem in Neural Machine Translation

Master's Thesis by Fabian Hirschmann February 2016

Referee: Prof. Dr. Johannes Fürnkranz
Supervisor: Jinseok Nam

TECHNISCHE UNIVERSITÄT DARMSTADT

Department of Computer Science
Knowledge Engineering Group

Addressing the Out-of-Vocabulary Problem in Neural Machine Translation

Submitted Master's thesis by Fabian Hirschmann

Referee: Prof. Dr. Johannes Fürnkranz
Supervisor: Jinseok Nam

Submission date: February 16th, 2016

Thesis Declaration

This Master's thesis has been carried out at the Knowledge Engineering in the Department of Computer Science, Technische Universität Darmstadt, Germany, under the guidance of Prof. Dr. Johannes Fürnkranz and Jinseok Nam.

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form. In the submitted thesis the written copies and the electronic version are identical in content.

Darmstadt, February 16th, 2016

_____

(Fabian Hirschmann)

Abstract

Traditional machine translation systems often require heavy feature engineering and the combination of multiple techniques solving different sub-problems. In recent years, several end-to-end learning architectures based on recurrent neural networks have been proposed. Unlike traditional systems, Neural Machine Translation (NMT) systems require minimal preprocessing and learn model parameters in a way of minimizing a pre-defined loss function. Due to both memory and time complexity when training such models, only a fixed number of words can be taken into account, which leads to the Out-of-Vocabulary (OOV) problem. In this work, we study simple yet effective approaches to alleviating the OOV problem in NMT. When using a manually created bilingual dictionary in conjunction with a language model, and a compound splitter, we obtain 2.88 BLEU points over a baseline on the German to English translation task. For English to German, we introduce target side compound splitting through a special syntax during training that allows the model to merge compound words, and gain 0.2 BLEU points. We also report results of translation hypothesis reranking and several preprocessing techniques from which we cannot achieve improvements.

Zusammenfassung

Traditionelle Systeme zur maschinellen Übersetzung benötigen oft erhebliches Feature Engineering und die Kombination mehrerer Methoden um verschiedene Teilprobleme während der Übersetzung zu lösen. In den letzten Jahren wurden mehrere Ende-zu-Ende Lernarchitekturen basierend auf Recurrent Neural Networks vorgeschlagen. Im Unterschied zu üblichen Systemen benötigen Neural Machine Translation (NMT) Systeme nur minimale Datenvorverarbeitung und die Modellparameter werden durch die Minimierung einer vordefinierten Verlustfunktion erlernt. Aufgrund der Speicher- und Zeitkomplexität beim Training dieser Systeme wird nur eine feste Anzahl an Wörtern in Betracht gezogen, woraus das Out-of-Vocabulary (OOV) Problem resultiert. In dieser Studie untersuchen wir ebenso einfache wie effektive Ansätze um die Auswirkungen des OOV-Problems zu begrenzen. Durch die Nutzung eines bilingualen und manuell erstellten Wörterbuchs in Verbindung mit einem Sprachmodell und einem Kompositazerleger erhalten wir eine Verbesserung von 2.88 BLEU Punkten verglichen mit einer Baseline bei der Übersetzung vom Deutschen ins Englische. Für die Übersetzungsfolge Englisch-Deutsch setzen wir Kompositazerleger für die Zielsprache durch eine spezielle Syntax während des Trainings ein, wodurch die Performanz um 0.2 BLEU Punkten steigt und das Modell zu Kompositaverschmelzung befähigt wird. Außerdem stellen wir Resultate für das Reranking von Übersetzungshypothesen und diversen Datenvorverarbeitungstechniken zur Verfügung, die allerdings nicht zu einer Verbesserung geführt haben.

Contents

1 Introduction

Machine Translation (MT) is a subfield of Computational Linguistics and is heavily influenced by Machine Learning. It has a long history that dates back to 1954 where researchers developed the Georgetown-IBM experiment that was able to translate sentences from Russian to English based on six grammar rules and a lexical dictionary of 250 items, which contained word stems and endings. At the beginning of this field, the focus was concentrated on the rule-based paradigm. This paradigm is based on morphological and syntactic rules and semantics of both the source and target language and is similar to manual word-to-word translations with only a bilingual dictionary at hand.

With the advances of computing power and a better understanding of the subject matter, the focus gradually shifted from rule-based translation systems to Statistical Machine Translation (SMT). The statistical approach has immensely improved the quality of machine translation over rule-based systems and was a giant step forward. Machine translation is not only an academic playing ground and has found its application in many commercial sectors. The most prominent consumer-facing form of this application is found in online translation services by Google and Microsoft that offer ad hoc translations. However, MT is also found in less open fields. For example, weather reports contain only a limited set of vocabulary and are often translated using machine translation. Likewise, MT is also experimented with in the medical field to translate medical reports written in a foreign language. Other examples include the professional translation industry, which uses machine translation as a way to increase productivity while ensuring quality by employing human translators as post-editors. The latter process is also known as Computer-Assisted Translation (CAT) and is based on the principle that MT is still not perfect.

1.1 Difficulties in Machine Translation

Machine translation has proven to be a difficult task for computer algorithms, and the reason is manifold. Before we go into these details and without prior experience in machine translation, it is trivial to imagine that for any source sentence to be translated, many different entirely correct translations exist. Given all possible German translations y for an English sentence x, the central principle behind SMT's decoding algorithm is to find the best English sentence that maximizes the probability of all possible sentences, i.e.:

$$y_{\text{best}} = \arg\max_{y} p(y|x) \tag{1.1}$$

One of the reasons that machine translation is hard is that natural language is highly *ambiguous*. When language is called ambiguous, it means that in can be understood in two or more possible ways or senses. Natural Language Processing (NLP) researchers often separate different kinds of ambiguities:

- Lexical ambiguity of a word or phrase means the word has more than one meaning. For example, considering the phrases *the back door* and *on my back*, the word *back* has two distinctive meanings. While humans are able to understand the meaning of a word depending on the context it is used in, this is much more difficult for a computer.
- Syntactic ambiguity may arise when one sentence has more than one syntactic structure (its syntax). For example, in the sentence *I shot an elephant in my pyjamas* can be understood in many different ways. Other popular examples of syntactic ambiguities are called *garden path sentences*. While the sentence *the old man the boat* is a grammatically correct sentence, it leads along a path that suddenly turns out not to work.
- Finally, semantic ambiguity happens when a sentence has more than one meaning. For example, the sentence *he saw the saw* can be understood either as a person cutting a saw or a person seeing a saw.

In computational linguistics, the research field of Word-Sense Disambiguation (WSD) tries to identify which sense of a word is used in a sentence, and is an open problem in NLP.

Not surprisingly, ambiguity is not the only problem in MT. Non-standard speech poses another problem especially for SMT and refers to the translation of language that is written in a non-standard form. An example of this may be found in colloquial speech or in online chats where the engaging parties often do not use proper word forms or punctuation. SMT systems rely on previously seen data that may or may not contain such non-standard language.

Even if all these problems are solved, the next difficult part is to evaluate the output of a translation system. Since we train our models on millions of example in SMT and test them thousands of sentences, one cannot assume that a human translator is able to assess the translation quality of thousands of sentences. Hence, we use bilingual text, i.e. text that was previously translated from one language to another, to train and test our models. However, we usually only have one translation for any given sentence in our training set available, but when our translation system comes up with an entirely different albeit perfectly correct translation, the quality assessment of an automatic evaluation metric will indeed turn out to be worse than what would be expected. Unfair evaluation is a serious problem that hinders the development of better machine translation techniques because researchers may deem a model that fits the test set better to be superior to a model that does not, even though the worse model may produce better results when evaluated by a human translator. As it turns out, machine learning evaluation is an active research field in itself and is also critical because translation systems can only be improved when we have the means to prefer one model over another. The way these metrics work, as well as three different metrics for evaluation, will also be discussed later in our work.

1.2 Thesis Motivation

As the use of Recurrent Neural Networks (RNNs) had become prevalent for SMT only recently, limited research on the issues unique to Neural Machine Translation (NMT) is available. One of the major issues with NMT not present in other designs such as phrase-based SMT and word-based SMT is the Out-of-Vocabulary (OOV) problem. In NMT, we are restricted to a finite set of vocabulary words that can be used in either the source or target language. Hence, we propose different strategies to alleviate the OOV problem. First, we try to depart from the common way

of building a vocabulary by just taking the n most frequent words in the training corpus, in lieu of only adding words that are neither named entities nor numbers. This idea stems from our previous analysis of unknown words, in which we found that many unknown words are in fact named entities. By doing so, we hope for the translation performance to increase because these words can be copied from the source sentence and not adding them to the vocabulary frees up precious space for words that cannot just be copied.

Our other strategies to counter the OOV problem are concerned with replacing unknown words in the translation output by querying the alignment model in the NMT attention mechanism. The alignment information of an unknown word in the target sequence is then used to find the aligned word in the source sequence, which in turn is used to come up with a suitable replacement for the unknown word in the target sentence. We try to find a replacement by copying the word from the source sequence, looking it up in a dictionary, or a combination of both. Dictionary replacement has been tried by other researchers at the same time, yet our approach differs in that we see the replacement as a probabilistic process in which the dictionary may return more than one candidate word.

When we inspected the translation output of an NMT model, we found that German compound words are often not translated into multi-word English phrases, and multi-word English phrases are hardly translated into a single German compound word. Hence, we try to mitigate this problem by employing both source and target side compound splitting.

In addition to the strategies above, we also try to increase the translation performance by optimizing the BLEU score. We call this process multi-pass decoding because an additional pass is inserted into the translation pipeline after the decoding step. This extra pass uses a machine learning model to rerank the candidate hypotheses that are outputted by the decoder by maximizing the predicted BLUE score. This process is well known in Automatic Speech Recognition (ASR) as well as phrase-based SMT but has not yet been applied to NMT.

1.3 Thesis Structure

We organize this Master's thesis as follows: In Chapter 2 we discuss the very recent advances in the field of machine translation that focus on neural networks as the underlying algorithm, a significant shift from the traditional SMT models or earlier word-based translation models. In Chapter 3, we provide a short introduction to the foundations of processing natural language using computers; this section is not meant to be a thorough introduction to this research field, and instead focuses only on the concepts that are of importance to the experiments we conducted. In Section 3.1, we also introduce the reader to Markov models by explaining sequence to sequence learning in another domain they are frequently used. Chapter 4 describes the basic concepts of SMT as well as several automatic evaluation metrics used to tune and compare models in SMT. Chapter 5 is an in-depth introduction to NMT, starting with the fundamentals of neural networks, followed by an overview and analysis of the different ways various researchers construct NMT systems. As NMT models are similar to hidden Markov models in that they accomplish sequence to sequence learning, we also briefly discuss the shortcomings of Markov models that were previously introduced as a component for natural language processing. In Chapter 6, we address the steps we have undertaken to preprocess our corpora and specifically talk about compound

splitting for the German language in Subsection 6.3.2. Finally, in Chapter 7 we provide a detailed analysis of the various experiments we have conducted during our work.

2 Related Work

Neural network based approaches have also been successful since an efficient way of learning word embeddings (Devlin et al. 2014). The major difference between such methods and NMT is that previously learned features from neural networks are used as extra information on top of existing translation systems, which are typically very complex and require heavy feature engineering. Contrarily, NMT models are end-to-end learning systems, which take an input sentence of arbitrary length and then generate an arbitrary length translation. Recently proposed neural machine translation models are often based on the encoder-decoder framework, in which a source sentence is encoded into a vector that is then decoded again into a sentence by the decoder. One of the challenges in NMT stems from the use of a fixed-sized vocabulary for both source and target language, which leads to unknown words being encoded as special *unknown* tokens. This issue is known as the Out-of-Vocabulary (OOV) problem and is specifically addressed in our work. We also explore different preprocessing techniques such as lowercasing and replacing named entities and numbers with special tokens as well as source and target side compound splitting.

Cho et al. (2014b) propose an encoder-decoder architecture in which they train the encoder and decoder jointly to maximize the conditional probability of the target sequence given the source sequence. The encoder of this approach encodes phrases as continuous space representations that preserve both the semantic and syntactic structure. The problem of NMT encoder-decoder frameworks is that its performance declines rapidly as longer sentences with more unknown words are translated (Cho et al. 2014a). Bahdanau, Cho, and Bengio (2014) propose a solution to this problem by extending the NMT framework in which an additional small feed-forward neural network soft-searches parts of the source sequence that are relevant for predicting the next word in the target sequence. The authors show that using this mechanism, the translation performance stays consistent as the sequence length increases. This soft attention mechanism additionally allows the model to learn word alignments jointly with translations, which is especially useful for postprocessing and the replacement of unknown words because an external word aligner is not required.

Sutskever, Vinyals, and Le (2014) also describe the general approach to sequence to sequence learning using recurrent neural networks. The input sequence is mapped to a fixed-length vector using Long Short-Term Memory units (Hochreiter and Schmidhuber 1997). This vector is then utilized in the decoder to decode the target sequence. The authors achieved state-of-the-art translation performance on the WMT14 English to French translation task. By reversing the source sequence, an improvement in translation performance was also obtained by the researchers.

In order to address the OOV problem, Jean et al. (2014) further extend the model by Bahdanau, Cho, and Bengio (2014) to hold a larger vocabulary without increasing training complexity, based on importance sampling. For English to French, their approach increases performance by 1.03 BLEU points (without ensembling or after-epoch training corpus reshuffling), however, for English to German, the translation performance worsens. Luong et al. (2015) address the OOV

problem by looking up unknown words in an automatically generated dictionary. As their Long Short-Term Memory (LSTM)-based model does not learn word alignments jointly with translation, they instead use an external word aligner to map words in the target sequence to words in the source sequence. Another key difference between our work is that the researchers build an automatically generated dictionary that produces one-to-one entries, whereas our approach is a probabilistic one in that the most likely word from the dictionary is selected.

Fritzinger and Fraser (2010) explore source and target side compound splitting for phrase-based SMT and propose a hybrid compound splitter that uses corpus statistics as well as a linguistic analyzer. They find that their approach outperforms purely corpus-driven approaches. Popović, Stein, and Ney (2006) also address compound words in SMT and find that German compound words are often not translated into multi-word English phrases, and multi-word English phrases are not translated to single token compound words in German. They further compare linguistics-based approaches to corpus-based approaches and deal with compound rejoining when German is on the target side.

There is plenty of literature available for translation hypothesis reranking in the context of phrase-based SMT. Och et al. (2004) propose minimum error rate training for re-ranking, L. Shen, Sarkar, and Och (2004) use ordinal regression to separate good translations from bad ones, and Duh and Kirchhoff (2008) use boosted models to improve performance over a log-linear model without additional features. As far as we are aware, no such literature for NMT is available yet.

3 Foundations of Natural Language Processing

Natural Language Processing (NLP) is a research field influenced by computer science, artificial intelligence, and computational linguistics dealing with the computational treatment of human language. In this chapter, we concentrate on the techniques used in our experiments that are mainly concerned with preprocessing of raw parallel text. The term parallel text refers to text that is written in more than one language. In our work, we use text that is written in English as well as German.

In this chapter, we make use of many special terms that we will now define briefly. In lexical analysis, breaking a sequence of text into words is known as *tokenization*, and a single element of the tokenizer's output is called a *token*. A token may be a word, punctuation, or special marker. It does not necessarily need to be a single word, because e.g. *New York* can be tokenized into a token consisting of two words. However, for the purpose of our work, we refer to words in a sequence of words as token due to the use of a tokenizer that does not support multi-word tokens.

A unit of lexical meaning disregarding the number of inflectional endings is called a *lexeme* and is the basic lexical unit of a language. The word *stem* is a part of a word, and in our work, it refers to a word to which affixes can be attached. The stem has different meanings in various contexts, i.e. the derivational suffix *-ship* can be connected to *friend* to form a new word *friendship*. But it can also refer to the part of the word that is common to all of its inflected variants, i.e. the stem for *friendships* is *friendship*. The latter meaning of stem is the one utilized by us. A related concept is a word's *lemma*, which is also a reduction from the word's inflectional form. However, while stemming employs a crude heuristic process to chop off the word endings, lemmatization tries to produce a correct word.

In Section 3.1, we will introduce the concept of Part of Speech (POS) tagging and assign each token a POS tag. This information is also used in our pre-training data analysis and preprocessing. POS tags are the formalized version of the word classes learned in middle-school. Examples of such tags include nouns, adverbs, and adjectives.

We also make use of Named Entities (NEs) during preprocessing after using a Named Entity Recognizer (NER). A NER finds NEs in a sentence, where a NE is an entity such as a person, place, or organization.

3.1 Part-of-Speech Tagging and Hidden Markov Models

A Part of Speech (POS) is a category of lexical items (e.g. words) that share similar grammatical properties. This most often refers to tagging words as nouns, verbs, adjectives, and other lexical categories. The categories used to tag words is known as the *tagset* and a POS tagger assigns

Table 3.1: Word having four different parts of speech.

Sentence	POS Tag of *back*
The back door	ADJ (adjective)
On my back	N (noun)
Win the voters back	ADV (adverb)
Promised to back the bill	V (verb)

each word in a sequence a tag from the tagset. For English, the Penn Treebank tagset (Marcus, Marcinkiewicz, and Santorini 1993) is commonly used while for German, the de-facto standard is defined by Schiller, Teufel, and Thielen (1995) as the Stuttgart Tagset (STTS).

Taggers are usually trained on a gold standard, i.e. they are trained on data that has previously been tagged manually by humans. As it is the case with MT, POS tagging is either done using a rule-based paradigm or statistical approach, although hybrid approaches are possible.

Lexical categories are sometimes divided into *open* and *closed* classes that refer to categories that commonly accept the addition of new words and categories where words are rarely added, respectively. Open classes often include nouns, verbs, adjectives and adverbs while prepositions, determiners, conjunctions and pronouns are closed classes. This distinction is important because it simplifies the development of taggers as determiners like *the* can easily be assigned the correct class.

This also leads us to the main challenge of POS tagging: There are multiple possibilities for assigning parts of speech. Consider the example sentences in Table 3.1 where the word *back* takes on four different part of speech tags. As intuition tells, the part of speech often depends on the *context* of a word, i.e. the words following and preceding it. Indeed, the context is a feature used in most POS taggers.

However, if using an algorithm that automatically models such sequences, keeping track of preceding words may not be required. One such model is known as an Hidden Markov Model (HMM) – a model in which the underlying system is assumed to be a Markov process of unobserved (hidden) states. This model was developed by Baum and Petrie (1966), although for the interested reader we recommend a tutorial written by Rabiner and Juang (1986).

We will now introduce the concept of an HMM using the simple example of POS tagging because later in our work we will discuss the similarites and differences between HMMs and RNNs for MT.

An HMM λ can be formalized as tuple $\lambda = (A, B, \pi)$. When having N states $S = S_1, \ldots, S_N$ and M different classes of observations $V = v_1, \ldots, v_M$ (POS tags), then $A \in \mathbb{R}^{N \times N}$ is the matrix of the state transition probabilities and its coefficients α_{ij} which specify the probability the HMM transitions from state S_i to state S_j. Let $B \in \mathbb{R}^{M \times N}$ be the matrix of emission probabilities and its coefficients $b_i(v_k)$ which hold the probability of class v_K being seen in state S_i. This concept is visualized in Figure 3.1. In this figure, the HMM can only reach adjacent states or the emitting state itself, but other configurations are possible. Figure 3.2 shows the trellis of an observation sequence of length 3 for the HMM in Figure 3.1

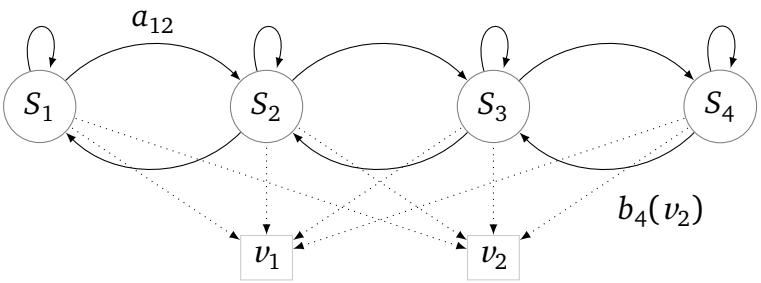

Figure 3.1: An HMM with 4 states which can emit 2 discrete symbols v_1 or v_2. a_{ij} is the probability to transition from state S_i to state S_j, while $b_j(v_k)$ is the probability to emit symbol v_k in state S_j.

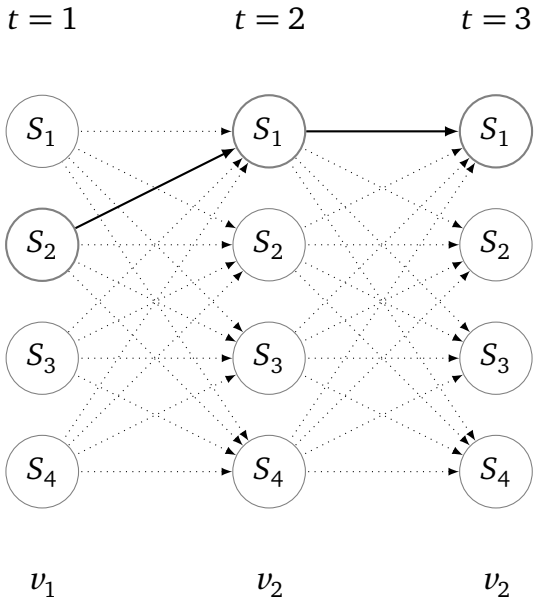

Figure 3.2: Trellis of the observation sequence v_1, v_2, v_2 for the above HMM. The thick arrows indicate the most probable transitions. As an example, the transition between state S_1 at time $t = 2$ and state S_4 at time $t = 3$ has probability $\alpha_2(1)a_{14}b_4(v_2)$, where $\alpha_t(i)$ is the probability to be in state S_i at time t.

Also, let $\pi \in \mathbb{R}^N$ be the starting probabilities which for POS can be as simple as the probability of starting a sentence with a specific class as determined by word frequency.

Then the *forward* part of the *Forward-Backward Algorithm* is defined as:

$$P(O|\lambda) \tag{3.1}$$

This is also known as the first problem of Rabiner and Juang (1986) and can be understood as the probability that an HMM λ produces a given observation $O = \{O_1, \ldots, O_T | O_k \in V\}$.

This term can be written as the sum over all probabilities over all possible state sequences that produce the given observations, multiplied with the probability of having passed through the state sequence of the given HMM:

$$P(O|\lambda) = \sum_{\text{all } Q} P(O|Q, \lambda) \cdot P(Q|\lambda) \tag{3.2}$$

Hence, both factors of the sum can be computed as:

$$P(O|Q, \lambda) = b_{q_1}(O_1) \cdot b_{q_2}(O_2) \cdot \ldots b_{qT}(O_T) \tag{3.3}$$

This is the probability of having observed O in the HMM λ, given a variable but fixed state sequence Q, multiplied with

$$P(Q|\lambda) = \pi_{q1} \cdot \alpha_{q_1,q_2} \cdot \alpha_{q_2,q_3} \cdot \ldots \cdot \alpha_{q_{T-1},q_T} \tag{3.4}$$

as probability of having passed through this state sequence using the HMM.

As it does not matter in which state the computation was at time $t-2$, it is enough only to take care of the states at t and $t-1$. Hence, the forward variables α are defined as:

$$\alpha_t(i) = P(O_1, O_2, \ldots, O_t, q_t = S_i|\lambda) \tag{3.5}$$

This is equivalent to the probability of being in state S_i after looking at the first t elements of the observation. The probability of being in a state after the first part of the observation has been processed is given by:

$$\alpha_1(i) = \pi_i \cdot b_i(O_1) \tag{3.6}$$

To calculate $\alpha_{t+1}(i)$ it is important to consider what can happen so that it leads to a state at $t+1$:

$$\alpha_{t+1}(j) = \left(\sum_{i=1}^{N} \alpha_t(i) \cdot \alpha_{i,j} \right) \cdot b_j(O_{t+1}) \tag{3.7}$$

That is, to get to $\alpha_{t+1}(j)$ for a given state S_j you could have been in any previous state S_i with a probability of $\alpha_t(i)$ and want to go to to the new state. The probability for this to happen is $\alpha_{i,j}$. The likelihood for making the current observation is $b_j(O_{t+1})$.

At some point the end of the observation is reached, and the overall probability of the observation is given by:

$$P(O|\lambda) = \sum_{t=1}^{N} \alpha_T(i) \tag{3.8}$$

This solves the first problem of Rabiner.

The next important question, also known as the second problem of Rabiner, is now of importance: Given an HMM λ and an observation sequence O, which is the most probable sequence Q the HMM has passed through:

$$\text{argmax } P(Q|O,\lambda) \tag{3.9}$$

This problem is solved using the *Viterbi* algorithm (Viterbi 1967). Let

$$\delta_t(i) = \max_{q_1,\ldots,q_{t-1}} P(q_1 q_2 \ldots q_t = i, O_1 O_2 \ldots O_t|\lambda) \tag{3.10}$$

be the set of probabilities of the most probable state sequence which were emitted by the first t observations ending in state S_i.

This set can be continued by induction for the next observations:

$$\delta_{t+1}(j) = \max_i(\delta_t(i)\alpha_{ij}) \cdot b_j(O_{t+1}) \tag{3.11}$$

The goal is to find the best probabilities for a given observation sequence until t. It consists of four steps, namely

- **Initialization**

$$\delta_1(i) = \pi_i b_i(O_1) \quad \text{for } 1 \leq i \leq N \tag{3.12}$$
$$\psi_1(i) = 0 \tag{3.13}$$

- **Recursion**

$$\delta_t(j) = \max_{1\leq i\leq N}(\delta_{t-1}(i)a_{ij}) \cdot b_j(O_t) \quad \text{for } 2 \leq t \leq T, 1 \leq j \leq N \tag{3.14}$$
$$\psi_t(j) = \arg\max_{1\leq i\leq N}(\delta_{t-1}(i)a_{ij}) \tag{3.15}$$

- **Termination**

$$P^* = \max_{1\leq i\leq N}(\delta_T(i))q_T^* = \arg\max_{1\leq i\leq N}(\delta_T(i)) \tag{3.16}$$

Where P^* is the highest probability of producing the observation with exactly one state sequence using the HMM, and q_T^* is the last state in this state sequence. Due to remembering which state led to going to q_T^* in $\psi_t(i)$, it is possible to look up the last state sequence that explains O. This brings us to the final part of the Viterbi algorithm, backtracking:

- **Back-Tracking**

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \quad \text{for } t = T-1, T-2, \ldots, 1 \tag{3.17}$$

Returning to the problem of POS tagging, consider the sentence

$$O = \{\text{I}, \text{live}, \text{near}, \text{live}, \text{animals}\} \tag{3.18}$$

in which the word `live` may have two different POS tags. Let the state transition probability matrix A be:

$$A = \begin{array}{c} \\ N \\ V \\ ADJ \\ P \\ PRO \end{array} \begin{array}{c} \begin{array}{ccccc} N & V & ADJ & P & PRO \end{array} \\ \begin{pmatrix} 0.1 & 0.5 & 0.1 & 0.3 & 0.2 \\ 0.3 & 0.0 & 0.2 & 0.3 & 0.2 \\ 0.5 & 0.1 & 0.2 & 0.1 & 0.1 \\ 0.4 & 0.1 & 0.3 & 0.1 & 0.1 \\ 0.1 & 0.7 & 0.1 & 0.1 & 0.0 \end{pmatrix} \end{array} \tag{3.19}$$

And the emission probability matrix B:

$$A = \begin{array}{c} \\ N \\ V \\ ADJ \\ P \\ PRO \end{array} \begin{array}{c} \begin{array}{cccc} I & live & near & animals \end{array} \\ \begin{pmatrix} 0.2 & 0 & 0 & 0.7 \\ 0 & 0.5 & 0 & 0.1 \\ 0 & 0.5 & 0.3 & 0.1 \\ 0 & 0 & 0.7 & 0.1 \\ 0.8 & 0 & 0 & 0 \end{pmatrix} \end{array} \tag{3.20}$$

After carrying out the forward algorithm described above in order to compute α_{ij} using Equation (5.65), we get

$$\alpha = \begin{pmatrix} 0.06 & 0 & 0 & 0 & 0.0016871 \\ 0 & 0.099 & 0 & 0.00143 & 0.0000396 \\ 0 & 0.015 & 0.00684 & 0.00396 & 0.0010788 \\ 0 & 0 & 0.02184 & 0 & 0.0000826 \\ 0.24 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.21}$$

According to Equation (3.8) the overall probability of the HMM λ to produce O is given by $P(O|\lambda) = 0.001917$. Analogously, performing the Viterbi algorithm yields:

$$\delta = \begin{array}{c} \\ N \\ V \\ ADJ \\ P \\ PRO \end{array} \begin{pmatrix} \\ 0.06 & 0 & 0 & 0 & 0.0009261 \\ 0 & 0.084 & 0 & 0.00882 & 0.0000265 \\ 0 & 0.012 & 0.00504 & 0.002646 & 0.0000530 \\ 0 & 0 & 0.01764 & 0 & 0.0000265 \\ 0.24 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.22}$$

The most probable sequence of POS tags can now be determined using ψ:

$$\{\text{PRO}, \text{V}, \text{P}, \text{ADJ}, \text{N}\} \tag{3.23}$$

The only important part missing to predict the POS tags is the question on how to approximate the coefficients for π, A, and B. This is also known of Rabiner's third problem. This problem can be solved using the Baum-Welch Algorithm (Welch 2003).

Modern POS taggers, not necessarily based on HMMs, are able to achieve performances of up to 97% (Manning 2011).

3.2 Language Models

Language Models (LMs) are an important component of machine translation models, in particular in phrase-based models. It measures how likely it is for a sequence of words to be uttered by a speaker. For any given input sequence $x = (x_1, \ldots, x_n)$, the function p_{LM} assigns a score to this input sequence in terms of a probability. This can be useful in many ways, i.e. phrase-based systems often output a large n-best list and a LM can be used to prefer the sentence *I like neural networks* over *like I neural networks*.

The currently most popular method for a language model is called an *n-gram* language model. It uses Markov chains to model the probability of a sentence appearing in a chunk of text, predicting one word at a time:

$$p(x_1, x_2, \ldots, x_n) = p(x_1)p(x_2|x_1)\ldots p(x_n|x_1, x_2, \ldots, x_{n-1}) \tag{3.24}$$

The probability is hence the product of words probabilities, where the history is usually limited to m words. The number m is known as the *order* of the model. A trigram model ($m = 3$) considers a two-word history to predict the third word, a bigram model ($m = 2$) considers one preceding word while a unigram model ($m = 1$) considers only the current word.

The probability for a trigram model is given by:

$$p(x_3|x_1, x_2) = \frac{\text{count}(x_1, x_2, x_3)}{\sum_w \text{count}(x_1, x_2, x)} \tag{3.25}$$

Hence, we simply count how often x_1, x_2 is followed by x_3.

In order to account for unseen words at prediction time, i.e. words that have not been seen in the training corpus, some smoothing is usually added to the model. The most commonly used smoothing method for n-grams models is known as *Kneser-Ney smoothing* by Kneser and Ney (1995). A modern implementation of such a model can be found in KenLM[1] (Heafield et al. 2013), and this is the model we are also using in our experiments.

Koehn (2009, Chapter 7) and Jurafsky and Martin (2000, Chapter 4) provide a thorough overview on n-gram language models. Just like as in machine translation, there is also research going on regarding LMs based on neural networks, and e.g. Bengio et al. (2003) or Schwenk (2007) proposed such models.

3.3 Named Entity Recognition

A Named Entity Recognizer (NER) is concerned with classifying tokens in a text to pre-defined categories such as persons, organizations, locations, etc. A NER is usually a machine learning algorithm trained on manually labeled data in which humans annotated words with NE categories. The tagset of such a recognizer is a finite set of categories, and common labels include *I-PER*, *I-LOC*, and *I-ORG*, referring to persons, locations, and organizations, respectively.

An exemplary sentence where a NER has been applied may look like:

$$\text{Jack}_{\text{I-PER}} \text{ worked for } \text{IBM}_{\text{I-ORG}} \text{ in } \text{Seattle}_{\text{I-LOC}}.$$

In our work, we use the Stanford NER[2] (Finkel, Grenager, and Manning 2005) for English, which is based on Conditional Random Fields (CRF). For German, we use the NER[3] developed by (Faruqui and Padó 2010). CRFs are statistical models for predicting structured data, and the conditional probability is defined over label sequences given a particular observation sequence. They are similar to HMMs, but they are conditional in nature, which leads to a relaxation of the strong independence assumptions of HMMs. CRFs were introduced by Lafferty, McCallum, and Pereira (2001). For more details on them please see the original paper or Wallach (2004).

The performance of a NER highly depends on the domain of the text to be classified. The highest performance is usually achieved on well-written news text, and fortunately, this is also the type of text we train our NMT models on. The scores for the Stanford NER on unseen news data are 88.21%, 87.68%, and 87.94% for precision, recall, and f-measure, respectively (Tjong Kim Sang and De Meulder 2003).

[1] https://kheafield.com/code/kenlm/

[2] http://nlp.stanford.edu/software/CRF-NER.shtml

[3] http://www.nlpado.de/~sebastian/software/ner_german.shtml

4 Statistical Machine Translation

Statistical Machine Translation (SMT) is the statistical approach to machine translation. It is based on the analysis and generation of prediction models that make use of bilingual text corpora, i.e. text that is written in more than one language simultaneously. It is so far the most successful approach and succeeds earlier methods such as rule-based machine translation. SMT works by detecting patterns in millions of documents that have previously been translated by human translators. It is important to note that SMT relies heavily on the availability of vast amounts of data, and a lack of such data means that no useful models can be produced.

Machine translation systems often use hybrid approaches by combining statistical and rule-based elements. For example, data may be preprocessed to guide the statistical engine, or the output of a statistical system may experience an application of rules to improve the overall quality. Translation systems that use only a rule-based engine at the core were seen at the beginning of this research field, and now statistical approaches dominate.

Early SMT systems were based on words as atomic units and one important component of these systems still used today is called an *aligner*. This component aligns a word from the source sentence with one or more words from the target sentence. As opposed to relying on a handcrafted dictionary, word alignments are learned through the analysis of bitext (bilingual text). A simple and naive approach would be to translate every word in the source sentence one by one. More sophisticated word-based models make use of probability theory to model conditionally words in a translation depending on the previously predicted words.

Phrase-based SMT models on the other hand view phrases as atomic units, thereby reducing the restrictions induced by word-based translation. They are currently the most successful and widely used models for translating into many languages. Unlike NMT, they rely on much knowledge and analysis of both the source and target language. For a comparison to NMT, we will introduce phrase-based SMT in the next section. An excellent textbook on these models is provided by Koehn (2009), on which the next section is also based.

4.1 Phrase-Based Machine Translation

As we have briefly mentioned, phrase-based models translate *phrases* as atomic units. These are also the models used by online services such as Google Translate[1], although Google works on neural network based translations systems as well (e.g. Sutskever, Vinyals, and Le (2014)).

The operation of a phrase-based model is depicted in Figure 4.1. In this illustration, the German input sentence is first segmented into a sequence of phrases, which are then translated individually into English sentences. The output phrases have also been reordered so that the verb follows the subject.

[1] `http://translate.google.com`

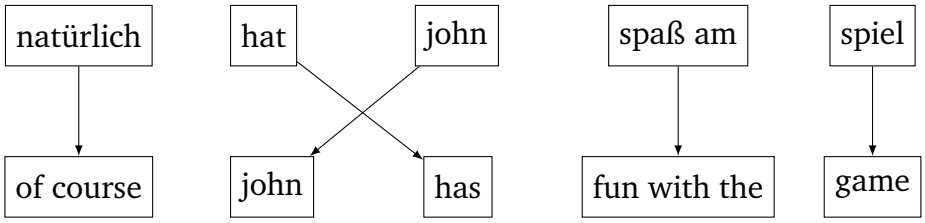

Figure 4.1: An illustration of phrase-based machine translation. The input phrases are translated into output phrases and possibly reordered.

The translations of the source phrases are looked up in a so-called *Phrase Translation Table*. The phrase translation table holds the probability of translating a source phrase to a target phrase, i.e. $\phi(y|x)$. For example, the likelihood of translating *natürlich* into *of course* may be $p(\text{of course}|\text{natürlich}) = 0.5$. The best translation y for an input sequence x is then simply given by

$$y_{\text{best}} = \arg\max_y p(y|x) \tag{4.1}$$

$$= \arg\max_y p(x|y)p_{LM}(y) \tag{4.2}$$

where $p_{LM}(y)$ is the score according to a language model as introduced in Section 3.2. The probability $p(x|y)$ can be further decomposed into

$$p(\bar{x}_1^I|\bar{y}_1^I) = \sum_{i=1}^{I} \phi(\bar{x}_i|\bar{y}_i)d(\text{start}_i - \text{end}_{i-1} - 1) \tag{4.3}$$

where ϕ is the phrase translation probability, d the reordering probability, and $\bar{x}$ are I source phrases. Reordering is performed by a *distance-based reordering model*, which can be seen as the number of words skipped when taking foreign words out of the sequence. Phrase-based models usually apply some decay to the probability d such that large movements are more expensive.

The phrase translation table ϕ can be built based on word alignments and together with the reordering model d and the language model $p_{LM}(y)$ forms a phrase-based log-linear SMT model. For a detailed explanation of how each component in this system is built, please refer to Koehn (2009).

Using this formulation, a probability score can be given for candidate translations given a source sentence. The *decoding* phase in phrase-based SMT is concerned with finding the best possible candidate. This problem is NP-complete (Knight 1999). Hence, a heuristic search algorithm such as beam search is used. Do note that beam search may not yield the best candidate translation, but an exhaustive search is impracticable due to NP-completeness.

4.2 Evaluation of Machine Translation Systems

Assessing the performance of the output of machine translation systems is a hard task, and many different automatic evaluation methods have been developed. As opposed to other fields of machine learning, no clear right or wrong answer can be provided, because human language comes in many correct forms. The major issues of MT evaluation include

- Language is variable: Many different correct translation exist
- Humans are subjective and may score a translation differently
- Which system is better: A or B?
- Target application: Domain specific language or general language?

Machine translation evaluation is a research topic itself, and an automatic method for evaluation is required in situations where human evaluation is too expensive, e.g. when determining the quality of different configurations of a MT system. These evaluation metrics are usually compared against human judgment scores through Pearson's or Spearman's correlation coefficients. Such correlation coefficients can be found in works by Machácek and Bojar (2014). In this Section, we will describe three of these metrics, BLEU, METEOR, and TER, and discuss their advantages and disadvantages in Subsection 4.2.4.

4.2.1 Bilingual Evaluation Understudy (BLEU)

The Bilingual Evaluation Understudy (BLEU) metric is an algorithm to compare the system output of a translation system to a human translation and was developed by Papineni et al. (2002). Essentially, BLEU measures the n-gram overlap between the machine translation output and one or more reference translations trough modified precision. For simplicity, in the following examples, we will concentrate on unigrams, but BLEU is nonetheless computed on n-grams.

Formally, precision is defined as:

$$p = \frac{\#\text{ of true positives}}{\#\text{ of true positives} + \#\text{ of of false positives}} \tag{4.4}$$

For machine translation evaluation, one would simply count up the number of candidate translation words which occur in the reference translation and then divide this number by the total number of words in the candidate translation.

However, in BLEU, a modification to the precision metric is introduced because MT systems tend to generate more words than necessary for a translation, and without this modification to precision a poor translation would yield a higher than reasonable score. Papineni et al. (2002) illustrate this problem using a toy example (Example 1). In this example, the unigram precision for the unmodified precision metric is 1, yet this translation does not make sense:

Example 1. Candidate: the the the the the the the.
Reference 1: The cat is on the mat.
Reference 2: There is a cat on the mat.

To counter this type of problem, BLEU first takes the maximum count of a word in any reference translations. Next, the total count of each candidate word is clipped by its maximum reference count. Finally, the clipped counts are summed up and divided by the total unclipped number of

candidate words (Papineni et al. 2002). This modified precision score is defined by the authors as:

$$p_n = \frac{\sum_{C \in \{\text{Candidates}\}} \sum_{\text{n-gram} \in C} \text{Count}_{\text{clip}}(\text{n-gram})}{\sum_{C' \in \{\text{Candidates}\}} \sum_{\text{n-gram}' \in C'} \text{Count}(\text{n-gram}')} \tag{4.5}$$

For obtaining the final BLEU score, the geometric mean of the modified n-gram precision for $n \in \{1, 2, 3, 4\}$ is built. These numbers were experimentally chosen by the authors in correlation with human judgments.

It is important to note that there are many variations of BLEU in existence, some of which are not documented very well and yet are widely used. Variations may include a type of BLEU score that does not lead to a 0 score in case there is no 4-gram match when computed on a sentence level, which is necessary for a scenario in which the translation is shorter than four words. This 0 score is due to BLEU computing the geometric mean of n-gram precisions, and having a 0 for any n-gram counts results in the score diminishing to 0 by multiplication with 0. Hence, a variant that takes this into account is known as smoothed BLEU.

The general practice in machine translation literature is to name the tool which was used for scoring, instead of just referring to it as BLEU score. Hence, our work uses the `multi-bleu.pl` Perl script[2] from Moses (Koehn et al. 2007). For Oracle scores, we use the BLEU scoring facilities of the `multeval` framework[3] (Clark et al. 2011). A crucial property of multeval's scoring services is the fact that it only works on lower-cased corpora; consequently, the score reported by this tool is different from case-sensitive Moses script.

4.2.2 Translation Error Rate (TER)

The Translation Error Rate (TER) quantifies the number of edits required for a translation to match a reference translation (Snover et al. 2006). It is hence provided as the minimum number of edits needed to a hypothesis such that it matches a reference exactly, normalized by the average length of the reference. It can be used with a single reference as well as multiple references and is defined as

$$\text{TER} = \frac{\#\text{ of edits}}{\text{average }\#\text{ of reference words}} \tag{4.6}$$

Edits possible in this metric are

- Insertions (insertion of a word into a sentence)
- Deletions (removal of a word from a sentence)
- Substitutions (replacement of a word in a sentence with another)
- Shifts (shift of a word in a sentence to a different position)

[2] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl
[3] `https://github.com/jhclark/multeval`

All of these edits share an equal cost.

This metric is conceptually similar to the Levenshtein distance, i.e. the number of edits required to match one string with another. The implementation we use is provided by Snover[4].

4.2.3 Metric for Evaluation of Translation with Explicit ORdering (METEOR)

Another more recent MT evaluation metric is given by Banerjee and Lavie (2005) and continued by Denkowski and Lavie (2014) and has implementations readily available[5]. The Metric for Evaluation of Translation with Explicit ORdering (METEOR) is more linguistically motivated and takes into account surface forms, stemmed forms, and meanings of unigrams. Based on a generalized concept of unigram matching between translation and reference, this metric computes a score using a combination of unigram precision and recall and a measure of fragmentation that reflects how well-ordered the words in the translation are in relation to the human-provided reference translation.

METEOR first runs an alignment algorithm which maps words in the translation of the system output. The space of these alignments is constructed by identifying all possible matches exhaustively according to the following matchers (Denkowski and Lavie 2014):

- **Exact:** Match identical words.
- **Stem:** Match identical stem using the Snowball Stemmer (Porter 2001).
- **Synonym:** Match words in case they are members in a synonym set according to WordNet (Miller et al. 1990).
- **Paraphrase:** Match phrases in case they are listed in a paraphrase table (Denkowski and Lavie 2010).

As this method may yield more than one alignment, alignment resolution is conducted using beam search given the heuristic described by Denkowski and Lavie (2014).

Each of the matchers mentioned above is assigned a weight and precision P and recall R is calculated using these weights. Then, the parameterized harmonic mean of P and R is calculated as:

$$F_{\text{mean}} = \frac{P \cdot R}{\alpha \cdot P + (1-\alpha) \cdot R} \tag{4.7}$$

METEOR also accounts for gaps and difference in word order between the reference translation and the system output by introducing a fragmentation penalty, which is defined as

$$\text{Pen} = \gamma \cdot \left(\frac{ch}{m}\right)^{\beta} \tag{4.8}$$

where m is the total number of matched words, averaged over the hypothesis and reference and ch is the number of chunks.

[4] http://www.cs.umd.edu/~snover/tercom/
[5] http://www.cs.cmu.edu/~alavie/METEOR/

The final METEOR score in then given by:

$$\text{Score} = (1 - \text{Pen}) \cdot F_{\text{mean}} \tag{4.9}$$

The parameters α, β, γ are tuning parameters and chosen to maximize correlation with human judgments.

In Figure 4.2 the METEOR scoring technique is visualized. In this figure, the reference translation is given by:

```
the first two victims , gravely wounded , were admitted to the
hospital , where the eighty year old man died .
```

Whereas the translation output of a MT system is:

```
the first two victims , seriously injured were admitted to the
hospital where the 80 year old has died .
```

The bullet marker (•) in the matrix correspond to exact matches of the word surface forms, while the circle marker (∘) indicate matches by other matchers. The color spectrum follows the word order in the reference translation.

In this example the words *gravely*, *wounded*, *eighty* are matched with *seriously*, *injured*, *80* as indicated by the circle marker. The second comma in the reference is not present in the hypothesis, and the word *has* in the hypothesis is not present in the reference. This is shown by the coloring and leads to fragmentation score of 0.439. The final METEOR score for this segment is 0.477.

4.2.4 Discussion on Evaluation Metrics

Automatic Evaluation metrics are plentiful, and some work well for one language while working poorly for another. Hence, we will briefly discuss the applicability of BLEU, METEOR, and TER to German and English. Machácek and Bojar (2014) provide an overview of 23 evaluation metrics from 12 research groups and evaluate these metrics against human judgment scores through Pearson correlation. These findings stem from the 2014 competition of the WMT Metrics Shared Task. In previous iterations of this contest, the Spearman correlation coefficient was used. However, Spearman's ρ converts metric scores and human scores to ranks and disregards the absolute difference. As Pearson's correlation coefficient does not suffer from this problem, the authors feel it provides a fairer evaluation method.

As part of the WMT14 Translation task, a large-scale manual annotation was performed to compare different MT systems. The participants were asked to rank the output sentences of these systems relative to each other. These human judgment scores were then collected by Machácek and Bojar (2014) and compared to the output of automatic evaluation metrics.

The results of this study can be seen in Table 4.1. The best performance for systems translating from German to English is given by METEOR while BLEU and TER share the same score. For the other way around, interestingly, the much simpler TER metric correlates much higher with human judgment than both METEOR and BLEU, with METEOR still taking a lead over BLEU. As

Table 4.1: System-level correlations of selected automatic evaluation metrics (Machácek and Bojar 2014)

Languages	Metric	Pearson's ρ
German → English	BLEU	0.952 ± 0.012
	METEOR	0.975 ± 0.009
	TER	0.952 ± 0.012
English → German	BLEU	0.216 ± 0.046
	METEOR	0.263 ± 0.045
	TER	0.324 ± 0.045

can be seen from this table, the correlation for translating into German is extremely low, but the authors note that this could be explained by a high number of participating systems of similar quality.

It is important to note that this study contains more than the three metrics we used in our work, and the highest correlation for German → English is 0.943 ± 0.020 and 0.357 ± 0.045 for the other way around. However, the often complex evaluation tools are not readily available, and most of the literature on machine translation provides scores in terms of BLEU, METEOR, or TER. Hence, we choose to report scores using the same metrics than other researchers.

To provide better translations, researchers tend to optimize for either metric, and we are no exception. This process does not come without risks, and A. Smith, Hardmeier, and Tiedemann (2014) conclude that sometimes optimizing reduces the translation quality. We try to counter this problem by manually inspecting the translation output. As we will see later in our work, some of our methods provide a modest increase in BLEU while introducing new capabilities to our models such as the ability to form target-side German compound words that would have otherwise simply resulted in an unknown token in the translation output. Also, by providing scores for three different metrics, the translation output is less likely to have deteriorated in quality if all three metrics improve despite optimizing for a single metric.

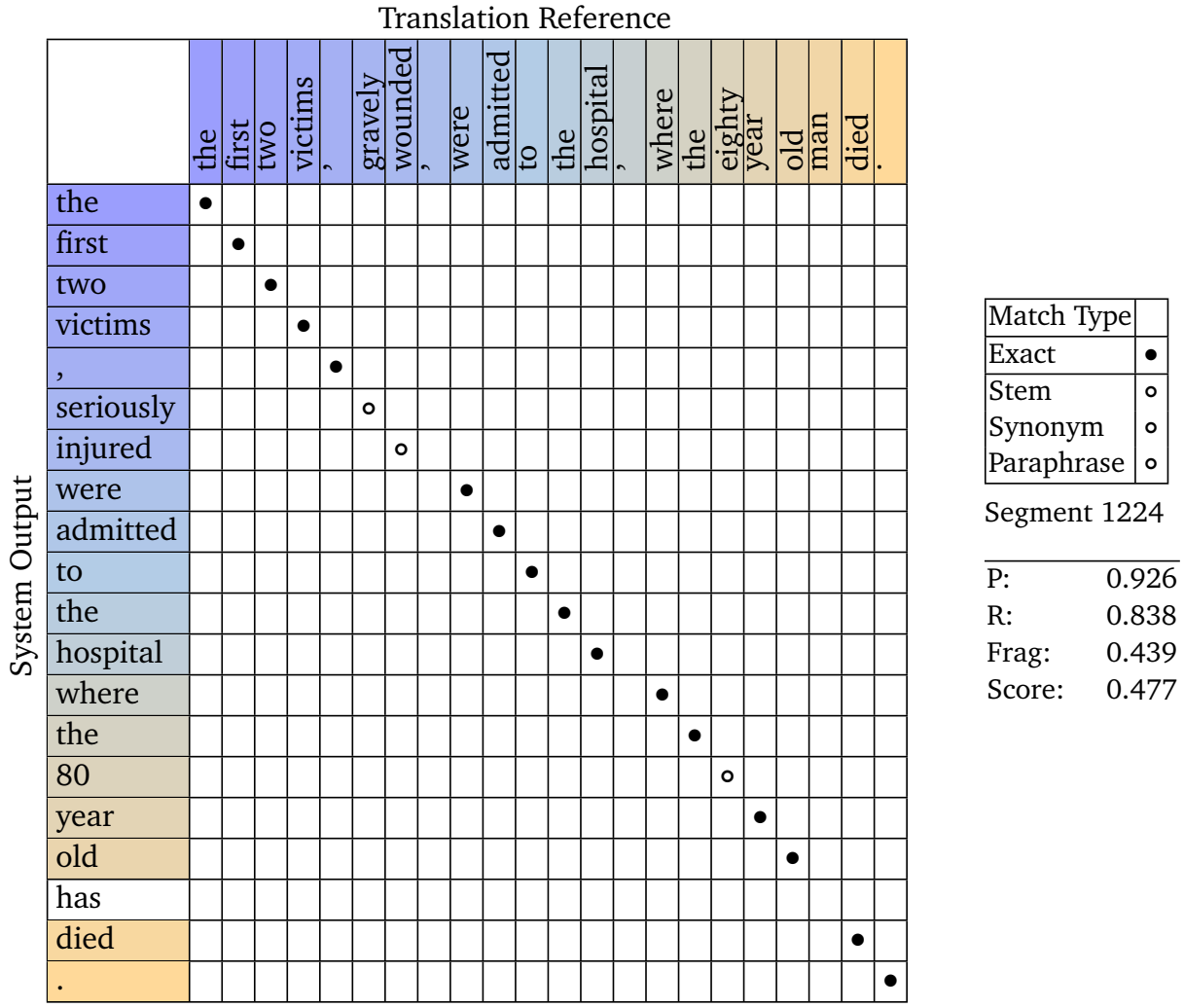

Figure 4.2: METEOR system output. The reference translation is given in the top row, whereas the MT system output is in the left column. The markers in the matrix represent word matches (exact, stemmed word, synonym word, paraphrased word). The color spectrum follows the reference word order.

5 Neural Machine Translation

In this Chapter, we describe a feed-forward Neural Network (NN) architecture in Section 5.1, followed by Recurrent Neural Networks (RNNs) in Section 5.2. These networks are trained using the backpropagation and the backpropagation through time algorithm, respectively. After discussing the exploding and vanishing gradient problem often experienced by RNNs, we explore advanced, recurrent architectures such as LSTM units and Gated Recurrent Units (GRUs). We continue onto introducing the reader to a variety of systems for language translation using RNNs. Neural Machine Translation (NMT) describes the concept of RNNs applied to the problem of SMT and we introduce various NMT frameworks, including designs proposed by Sutskever, Vinyals, and Le (2014), Cho et al. (2014b) and Bahdanau, Cho, and Bengio (2014).

The section on NNs is based on Bishop (2006) and Russell (1995) as well as the Unsupervised Feature Learning and Deep Learning Tutorial by Stanford University (Ng et al. 2015). The part on RNNs is based on Goodfellow, Courville, and Bengio (2015) and LeCun, Bengio, and Hinton (2015) while the section on NMT is based on the respective papers as well as Cho (2015) and Britz (2015).

5.1 Foundations of Neural Networks

In machine learning, NNs are a family of algorithms which approximate target functions for given inputs. They are inspired by biological neural networks and have their origins in mathematical representations of information processing in biological systems (McCulloch and Pitts 1943; Widrow and Hoff 1960; Rosenblatt 1962; Rumelhart, Hinton, and R. Williams 1986).

In a NN, artificial nodes (also known as *units* or *neurons*) are connected using directed links to solve problems such as classification or regression. These neurons are devices with a variety of inputs and a single output.

The elementary building block of a neural network is called a *neuron* or *unit*. One such unit is connected to other units using directed *links* and a numeric *weight* $w_{i,j}$ which determines the strength between neuron i and neuron j. Each neuron also has a dummy input $a_0 = 1$ with a corresponding weight $w_{0,j}$ (the bias). A neuron is characterized by computing the weighted sum of its inputs:

$$z_j = \sum_{i=0}^{n} w_{i,j} a_i \tag{5.1}$$

The concept of a neuron is visualized in Figure 5.1, in which the neuron has three input links (plus one dummy link with a value of 1) and three output connections. After applying the activation function f, the output of this unit is given by:

$$a_j = f(z_j) = f\left(\sum_{i=0}^{n} w_{i,j} a_i\right) \tag{5.2}$$

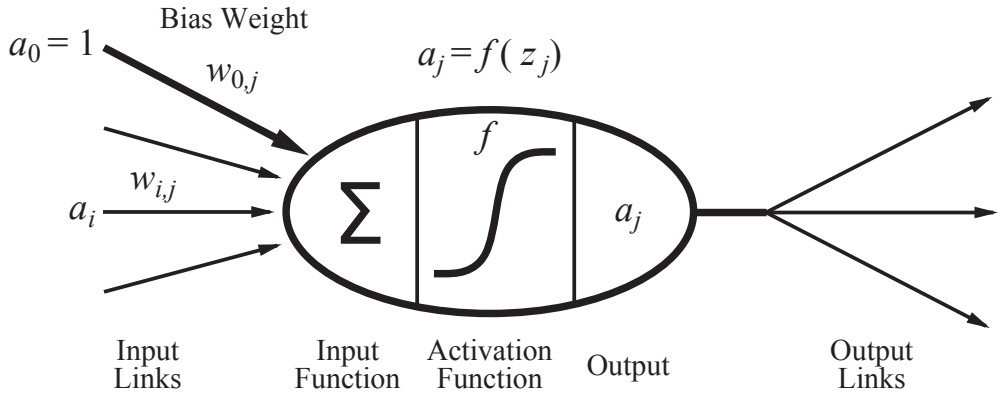

Figure 5.1: Depiction of a neuron by Russell (1995, p. 727). The output activation is given by $a_j = f(z_j) = f(\sum_{i=0}^{n} w_{i,j} a_i)$, where $w_{i,j}$ is the weight of the link from unit i to this unit and a_i is the output activation of unit i.

The activation function is what gives the neural networks its nonlinear capabilities and should have the following properties:

- **Nonlinear**: To capture nonlinear relationships between input variables, it is highly advisable to use a nonlinear activation function.
- **Differentiable**: To use gradient-based optimization methods, the activation function needs to be continuously differentiable.
- **Monotonic**: The error surface regarding a single-layer model is guaranteed to be convex if the activation function is monotonic.

A popular class of activation functions is called sigmoid, whose members are characterized by a characteristic S-shape. One of the most natural functions here is the logistic function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.3}$$

With its derivative:

$$\sigma'(x) = \frac{e^x}{(e^x + 1)^2} = \sigma(x) \cdot (1 - \sigma(x)) \tag{5.4}$$

As shown in Figure 5.2 (right) this function is monotonically increasing and asymptotes as $\pm\infty$ is approached. However, it is not symmetric with respect to the origin, i.e. it changes when reflected across both the horizontal and vertical axes.

Another activation function is the hyperbolic tangent (tanh) function:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.5}$$

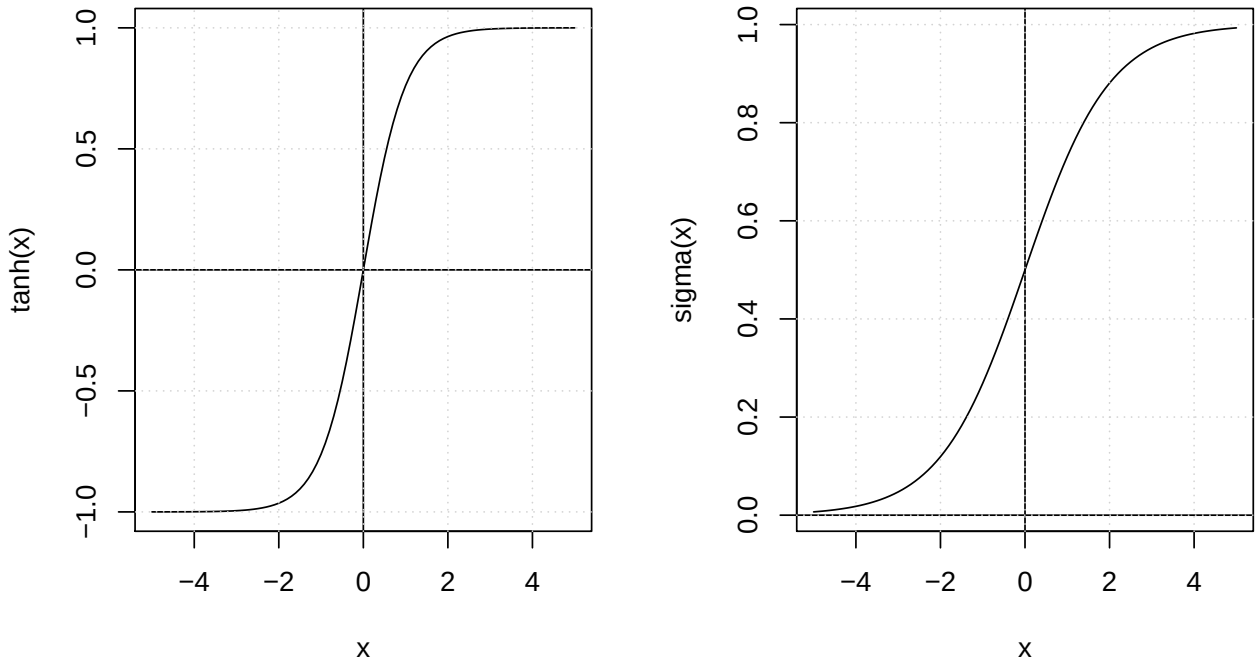

Figure 5.2: Plot of common neural network activation functions. Tanh (left) is symmetric with respect to the origin while sigmoid (right) is not.

With its derivative:

$$\tanh'(x) = \frac{4e^{2x}}{(e^{2x}+1)^2} = 1 - \tanh(x)^2 \tag{5.6}$$

Which is related to tanh by the following linear transformation:

$$\tanh(x) = 2\sigma(2x) - 1 \tag{5.7}$$

A plot of this functions can be found in Figure 5.2 (left). The tanh function is similar to the standard logistic function, yet it is symmetric about the origin. Often, the choice of an activation function matters for technical reasons, i.e. optimization. The tanh function should be preferred over the asymmetric sigmoid function because it often converges faster (LeCun et al. 2012). In our work on language translation using Neural Networks, the tanh function is used.

To create a neural network, these single neurons are interconnected so that the output of one neuron can be the input of another. Neural networks are usually created in layers, and the first layer is called the *input layer* while the last layer in the network is referred to as the *output layer*. Any layer between the input layer and the output is known as *hidden layer* because its values are not observed in the training data. The bias is omitted in this figure.

Many different layouts for neural networks have been developed, but one important distinction regarding their configuration can be made:

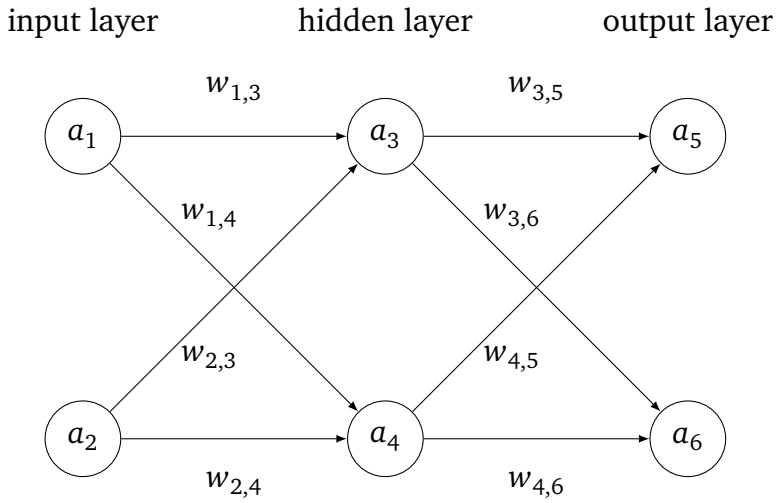

Figure 5.3: Exemplary depiction of a neural network with two nodes in each layer.

- **Cyclic:** Networks with cycles such as RNNs.
- **Acyclic:** Networks without cycles are referred to as Feedforward Neural Networks (FNNs).

An exemplary FNN is given in Figure 5.3. To calculate the output of this network, the units in the input layer are set to the value according to the training data:

$$(a_1, a_2) = (x_1, x_2) \tag{5.8}$$

Any subsequent outputs can then be calculated as a function of the inputs and weights. For instance, the first output node's value (a_5) is given by:

$$\begin{aligned} a_5 &= f(w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4) && (5.9)\\ &= f(w_{0,5} + w_{3,5}f(w_{0,3} + w_{1,3}a_1 + w_{2,3}a_2) + w_{4,5}f(w_{0,4} + w_{1,4}a_1 + w_{2,4}a_2)) && (5.10)\\ &= f(w_{0,5} + w_{3,5}f(w_{0,3} + w_{1,3}x_1 + w_{2,3}x_2) + w_{4,5}f(w_{0,4} + w_{1,4}x_1 + w_{2,4}x_2)) && (5.11) \end{aligned}$$

This is called the *forward propagation* step in neural networks because the values of the units are calculated starting with the neurons in the input layer and then any subsequent layer.

Note that some authors prefer to model the bias explicitly, i.e. $a_j = \left(\sum_{i=1}^{n} w_{i,j}a_i + b_j\right)$ which complicates the introduction of neural networks slightly.

Now that we know how to calculate the output of each neuron in a neural network, we need an algorithm to train the parameters of this system. A standard method of training neural networks used in conjunction with an optimization method is called *backpropagation* (Rumelhart, Hinton, and R. Williams 1986). This method calculates the gradient of a *loss function* jointly with the parameters of the network by alternately passing forwards and backward updates through the network.

Our exemplary network has multiple output nodes, and a scalar function $h_w(x) = a_n$ for each output node n is inconvenient. A much more convenient representation for our network is to use

a vector function $h_W(x) = (a_5, a_6)$ where a_5 and a_6 are nodes in the output layer in Figure 5.3. This is important in multi-layer networks where both a_5 and a_6 depend on all the input-layer weights and updates depend on errors in both a_5 and a_6. For loss functions that are additive across the components of the error vector $\mathbf{y} - \mathbf{h_W(x)}$, this is simple. In this example, we want to minimize the squared loss given by

$$\frac{\partial}{\partial w}\text{Loss}(\mathbf{w}) = \frac{\partial}{\partial w}|\mathbf{y} - \mathbf{h}(\mathbf{w})|^2 = \frac{\partial}{\partial w}\sum_k (y_k - a_k)^2 = \sum_k \frac{\partial}{\partial w}(y_k - a_k)^2 \tag{5.12}$$

The leftmost term is just the gradient of the loss corresponding to the k^{th} output, computed as if other outputs did not exist. The chain rule comes in handy for this calculation. Genereally speaking, the chain rule can be used for computing the derivate of the composition of two or more functions. Let $x = x(t)$ and $y = y(t)$ be differentiable at t and $z = f(x, y)$ at $(x(t), y(t))$. Then $z = f(x(t), y(t))$ is differentiable at t and:

$$\frac{dz}{dt} = \frac{\partial z}{\partial x} \cdot \frac{dx}{dt} + \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial t}. \tag{5.13}$$

The reason this is called the chain rule is that it can be applied sequentially to many nested functions. The non-zero weights $w_{j,k}$ that connect to the k^{th} output units can now be computed as:

$$\frac{\partial \text{Loss}_k}{\partial w_{j,k}} = -2(y_k - a_k)\frac{\partial a_k}{\partial w_{j,k}} = -2(y_k - a_k)\frac{\partial f(z_k)}{\partial w_{j,k}} \tag{5.14}$$

$$= -2(y_k - a_k)f'(z_k)\frac{\partial z_k}{\partial w_{j,k}} = -2(y_k - a_k)f'(z_k)\frac{\partial}{\partial w_{j,k}}\left(\sum_j w_{j,k}a_j\right) \tag{5.15}$$

$$= -2(y_k - a_k)f'(z_k)a_j = -a_j\Delta_k \tag{5.16}$$

Where $\Delta_k = \text{Err}_k \times f'(z_k)$ is the modified error, and Err_k the k^{th} component of the error vector $\mathbf{y} - \mathbf{h_w}$, and z_k is the same than in Equation (5.1). We can now expand out the activations a_j and reapply the chain rule:

$$\frac{\partial \text{Loss}_k}{\partial w_{i,j}} = -2(y_k - a_k)\frac{\partial a_k}{\partial w_{i,j}} = -2(y_k - a_k)\frac{\partial f(z_k)}{\partial w_{i,j}} \tag{5.17}$$

$$= -2(y_k - a_k)f'(z_k)\frac{\partial z_k}{\partial w_{i,j}} = -2\Delta_k\frac{\partial}{\partial w_{i,j}}\left(\sum w_{j,k}a_j\right) \tag{5.18}$$

$$= -2\Delta_k w_{j,k}\frac{\partial a_j}{\partial w_{i,j}} = -2\Delta_k w_{j,k}\frac{\partial f(z)}{\partial w_{i,j}} \tag{5.19}$$

$$= -2\Delta_k w_{j,k}f'(z_j)\frac{\partial z_j}{\partial w_{i,j}} \tag{5.20}$$

$$= -2\Delta_k w_{j,k}f'(z_j)\frac{\partial}{\partial w_{i,j}}\left(\sum_i wi, ja_i\right) \tag{5.21}$$

$$= -2\Delta_k w_{j,k}f'(z_j)a_i = -a_i\Delta_j \tag{5.22}$$

Calculating the error in the output layer is easy, i.e. it is just the difference between the network's output and the observed value, i.e. $\mathbf{y} - \mathbf{h_W}$. For hidden layer, it is much more complicated. Here, we need to use the concept of *backpropagation* where the error is propagated back starting from the output layer. Let Err_k be the k^{th} component of the error vector $\mathbf{y} - \mathbf{h_W}$. Also let the modified error be $\Delta_k = \text{Err}_k \times f'(z_k)$. Then, the weight update rule is given by

$$w_{j,k} \leftarrow w_{j,k} + \alpha \times a_j \times \Delta_k \tag{5.23}$$

where α is a small value called the *learning rate*.

The idea to update the weights between the input units and hidden units is that the hidden node j is in part responsible of the error Δ_k in each of the output nodes it links to. Hence, the values for Δ_k are divided with regard to the strength of the link between the hidden node and the output node. The error then propagated back to the hidden layer to provide Δ_j. This gives us the following propagation rule:

$$\Delta_j = f'(z_j) \sum_k w_{j_k} \Delta_k \tag{5.24}$$

And the update rule:

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta_j \tag{5.25}$$

The backpropagation algorithm can be reduced to a few simple steps:

- Compute the error values Δ for the output units as the difference between the observed value and the predicted value
- Repeat until the first hidden layer is reached, starting with the output layer:
  - Propagate the error values Δ back to the previous layer
  - Update the weights $w_{i,j}$ between node i and j

Once the error term for each node (except the nodes in the input layer) is calculated, the weights of the network are updated and these forwards and backward updates are alternately repeated until a stopping criterion is satisfied.

5.2 Recurrent Neural Networks

A Recurrent Neural Network (RNN) as developed by Rumelhart, Hinton, and R. Williams (1986) extends the concept of a simple NN such that connections between units form a directed cycle allowing for dynamic temporal behavior. The network has an implicit internal memory which is specially useful when processing sequences.

In the basic architecture of this network, each node has a directed connection to every other node. However, this is not the only topology an RNN can inherit. Figure 5.4 illustrates three different topologies for neural networks. On the left, we have a traditional FNN where connections are only going forward. The network in the middle is a layered network with an input layer, a

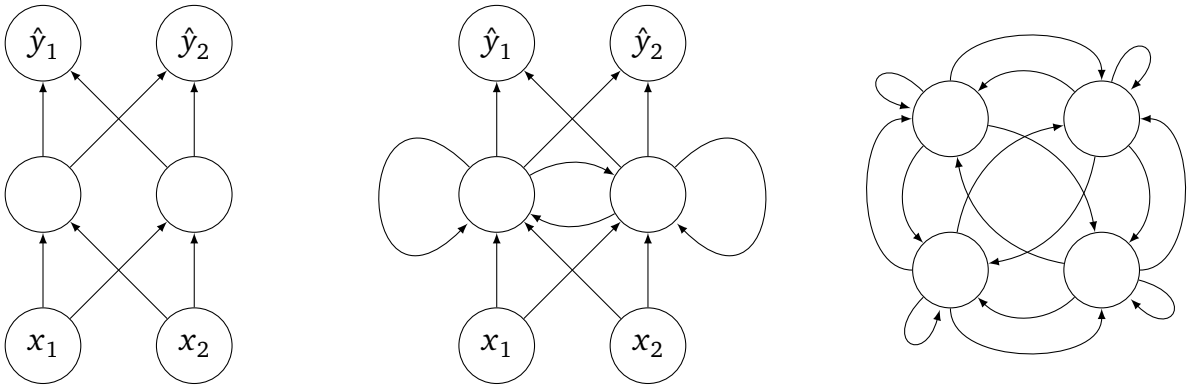

Figure 5.4: Different topologies for neural networks: (left) Feed-forward neural network. (middle) Network with fully recurrent hidden layer. (right) Fully connected recurrent network.

Figure 5.5: Illustration of a neural network in which layers are depicted as single objects. (left) Feed-forward network with one hidden layer that corresponds to the left illustration in Figure 5.4. (right) Recurrent network with one fully-connected hidden layer corresponding to Figure 5.4 (middle).

fully recurrent hidden layer, and an output layer. The network on the right is a fully connected recurrent network in which every node has a connection to every other node. For simplicity, we can illustrate the layers in a neural network as single objects as shown in Figure 5.5. This simplification will be useful later when we explain the temporal behavior of RNNs. The important part to understand about RNNs is that they lift the artificially imposed restriction of FNNs to allow only connections going forward.

In traditional neural networks, we assume that all inputs and outputs are independent of each other. However, for language translation any many other applications, this does not hold true. When translating a sentence from English to German, the n^{th} word may depend on the $(n-1)^{\text{th}}$ word. Consider the sentence

```
The man who lives in New York has died.
```

The word *died* at the end of the sequence relates to the word *man* at the beginning of the phrase, while the word *York* goes together with *New* to form *New York*. In the latter case, the word *New* depends on the following word and disregarding this fact leads to an entirely different meaning of the word *New*. In fact, for the latter case, we need to look forward in time, which can be accomplished by a Bidirectional Recurrent Neural Network (BRNN). However, right now we concentrate on introducing RNNs that can only look backward in time and explain BRNNs later in this Section.

An RNN does not make this assumption of independence because it performs the same computation for every element in a sequence and the output of a unit depends on the previous computations. This concept is visualized in Figure 5.6 in which the RNN from Figure 5.5 (right) is *unrolled* or *unfolded* into a full network. This technique of unrolling refers to writing out

the computations required for the complete sequence at hand similar to what can be done to a sequence of three elements in a for-loop, and the result looks much like a simple FNN. In this figure x_t refers to the observation at time t, i.e. for NMT this is the 1-of-V encoded word built from the vocabulary V for the t^{th} in the source sentence. The weight matrices need to be adjusted a little bit for RNNs:

- U: Input layer → hidden layer. Similar to the one used in FNNs.
- V: Hidden layer → output layer. Specifies how much importance to accord for present inputs.
- W: Previous hidden layer → hidden layer. Specifies the importance for past hidden states.

Where $a \rightarrow b$ denotes the weight from a to b. The error they produce will return via backpropagation, and their values are iteratively adjusted. The output is denoted by $\hat{y}$. The hidden states $s_t = f(Ux_t + Ws_{t-1})$ forms the memory of the network because its calculation is based on the previous hidden states. The activation function f in our case is tanh, defined by Equation (5.5). The output $\hat{y}_t$ at step t is used to predict e.g. the next word in a sequence using probabilities across the vocabulary. So, if we want to predict the next word when translating from one language to another, the result would be a vector of probabilities across our vocabulary, i.e. $\hat{y}_t = \text{softmax}(Vs_t)$. A thorough introduction to NMT will be given in later in this chapter.

The important thing to note here is that as opposed to traditional NNs, the parameters U, V, and W are shared across all computations so that the same task is performed at each step. Sharing is specifically useful for NMT in which dependencies between words exist. Sharing is also unlike traditional NNs in which the network's internal state (the parameters) is computed disregarding any preceding words in the sentence.

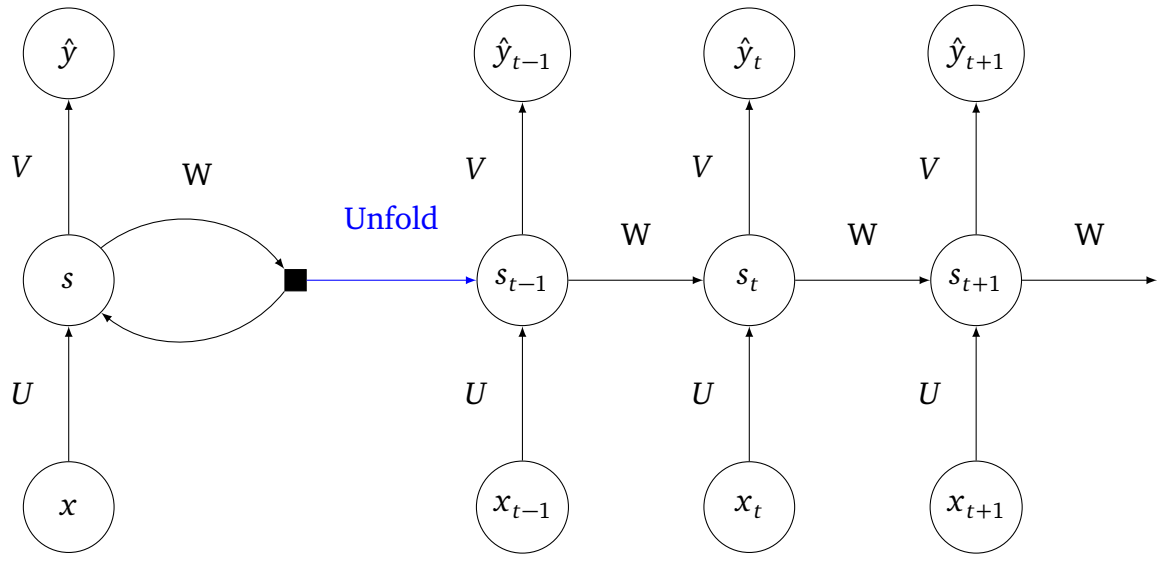

Figure 5.6: The unfolding in time of the computation in the forward computation of a RNN. The weight matrices U, V, W are the weights between the corresponding layers, x_t is an observation sequence at time t, $s_t = f(Ux_t + Ws_{t-1})$ are hidden states at time t, and $\hat{y}_t$ is the outpout of this network.

In order to train an RNN the backpropagation algorithm is used, just like for simple NNs. However, due to the parameters being shared by all time steps in the network, a modification

is necessary. As the gradient at each output depends on the current as well as all previous time steps, an algorithm named Backpropagation Through Time (BPTT) was developed. This algorithm performs Stochastic Gradient Descent (SGD) on a whole unfolded network. BPTT is very similar to standard backpropagation, except that the gradients for W need to be summed up at each time step due to the parameters being shared.

Given an input sequence $x = \{x_1, \ldots, x_T\}$ and an observed sequence $y = \{y_1, \ldots, y_{T'}\}$, $s_t = \tanh(Ux_t + Ws_{t-1})$, and $\hat{y}_t = \text{softmax}(Vs_t)$, the total loss for an example in the dataset is then given by the sum of losses over all time steps, i.e.:

$$L_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t \tag{5.26}$$

$$L(y, \hat{y}) = -\sum_t L_t(y_t, \hat{y}_t) = -\sum_t y_t \log \hat{y}_t \tag{5.27}$$

The gradients of the error with respect to U, V, W can then be calculated by

$$\frac{\partial L_t}{\partial V} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial V} \tag{5.28}$$

$$= \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial Vs_t} \cdot \frac{\partial Vs_t}{\partial V} \tag{5.29}$$

$$= (\hat{y}_t - y_t) \otimes s_t \tag{5.30}$$

where $\otimes$ is the outer product. As shown by this calculation, $\frac{\partial L_t}{\partial V}$ only depends on values of the current time step t. The gradient for V can then be obtained using matrix multiplication. Considering $\frac{\partial L_t}{\partial W}$ and $\frac{\partial L_t}{\partial U}$, we can see that they not only depend on the current time step, but also on the previous ones:

$$\frac{\partial L_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial s_t} \cdot \frac{\partial s_t}{\partial \boxed{s_k}} \cdot \frac{\partial \boxed{s_k}}{\partial W} \tag{5.31}$$

For $k < t$ the gradient depends on the previous states, as highlighted by the boxes. Hence, we need to backpropagate gradients from $t = T$ with T being the latest time step to $t = 0$ by adding the contributions. This concept is shown in Figure 5.7 in which the gradients at time $t = 3$ are backpropagated all the way to $t = 0$.

Going back to our example sentence earlier in this Section, we have seen that often an RNN needs to look forward in time in addition to looking backward. It turns out that we can stack two RNNs with the final output computed based on the hidden states from both networks, processing the sequence from left to right using the first network and from the right to the left using the second network, finally combining the two for unified output. This concept is known as a Bidirectional Recurrent Neural Network (BRNN) and was introduced by Schuster and Paliwal (1997).

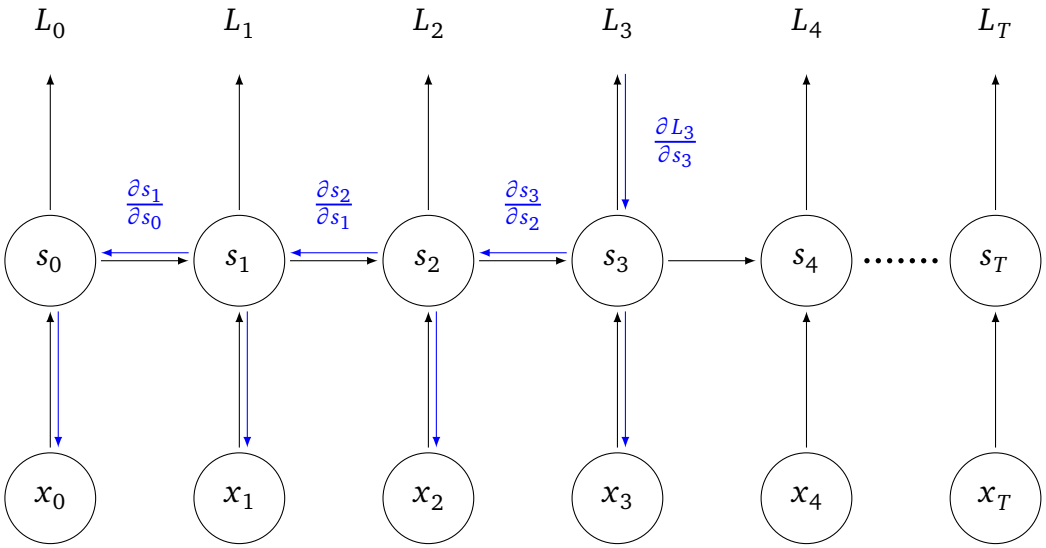

Figure 5.7: Backpropagation of the gradients in the BPTT algorithm. In this example, the gradients from $t = 3$ are backpropagated to $t = 0$.

5.2.1 Vanishing and Exploding Gradient Problem

The vanishing gradient problem is a problem affecting NNs with gradient-based learning methods and backpropagation and was first discovered by Hochreiter (1991) and further explored by Bengio, Simard, and Frasconi (1994). As each weight receives an update proportional to the gradient of the loss function with respect to the current weight, activation functions such as tanh have gradients in the range of $(0, 1)$. Since computation is done using the chain rule, many of these numbers are multiplied together, and due to the way the computation is done it means that the gradient decreases exponentially as it is backpropagated.

The vanishing gradient problem leads to RNNs having difficulties learning *long-range dependencies* up to a point where it gets impossible for the model to learn correlations between temporally distant events. Reiterate our example English sentence:

```
The man who lives in New York has died.
```

The key observation here is that the man has died, yet the words *man* and *died* are very far apart. The RNN will hence have difficulties modeling the long-range dependency between *man* and *died*. Equation (5.31) computes the gradient being backpropagated. Note that $\frac{\partial}{\partial s_k} s_t$ corresponds to the chain rule, i.e.:

$$\frac{\partial L_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial s_t} \cdot \boxed{\frac{\partial s_t}{\partial s_k}} \cdot \frac{\partial s_k}{\partial W} \tag{5.32}$$

This gradient can be rewritten as

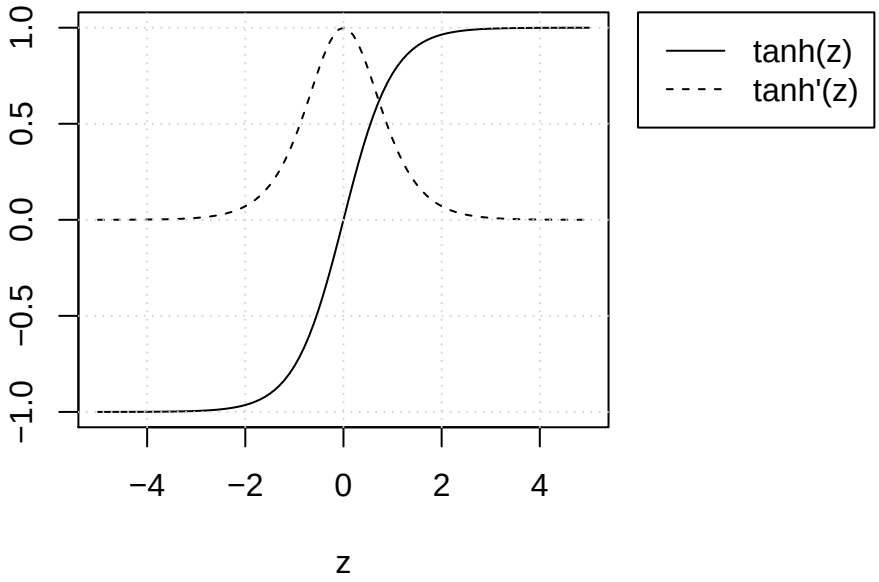

Figure 5.8: Plot of tanh and its derivative.

$$\frac{\partial L_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial s_t} \cdot \left(\prod_{j=k+1}^{t} \frac{\partial s_j}{\partial s_{j-1}} \right) \cdot \frac{\partial s_k}{\partial W} \tag{5.33}$$

due to the result being a matrix (the Jacobian matrix) whose elements are all the point-wise derivatives (see Pascanu, Mikolov, and Bengio (2013)). The 2-norm of the Jacobian matrix has an upper bound of 1, because as visualized in Figure 5.8 both tanh and its derivative have an upper limit of 1. You can also see that tanh has a derivative approaching 0 as $\pm\infty$ is approached. When approaching 0 the neurons become saturated, and the gradient values are shrinking exponentially, eventually vanishing altogether, i.e. becoming 0. This problem is also reinforced by the standard approach of initializing weights, i.e. by choosing weights using a Gaussian with mean 0 and standard deviation 1.

Going back to our example sentence, this means that the gradient contribution of the word *man* may become 0 by the time we are dealing with the word *died*. The gradient being tiny and becoming 0 is problematic when we have dependencies between words.

Depending on the activation function used, instead of the gradients becoming tiny, they may become enormous. And indeed, this is known as the *exploding gradient problem*. This problem can easily be constructed by choosing very large weights and biases so that $f'(Ux_t + Ws_{t-1} + b)$ is not too small.

There are many attempts to prevent these two problems, and some have only been proposed recently. These techniques include:

- Good initialization of the weight matrix W (Le, Jaitly, and Hinton 2015).
- Use of a rectifier linear unit (ReLU) activation function (Le, Jaitly, and Hinton 2015), i.e.

$$f(z) = ln(1 + e^z) \tag{5.34}$$

  which has a derivative that equals the logistic function that is not bounded by 1.
- Use of LSTM (variant), which will be described in the next Section.

Research seems to go currently into the direction of using more sophisticated recurrent hidden units like LSTMs or one of its variants. These are also the units we used in our work

5.2.2 Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU)

In the previous Section, we have seen how a vanilla RNNs tend to lose their memory as we move further along the input sequence due to the vanishing gradient problem. An LSTM network or GRU network brings back the memory the vanilla RNN lost. They are a particular kind of RNN and can learn long-term dependencies. Learning these dependencies makes them well-suited for many tasks that require the model to have memory, and indeed, they seem to outperform RNNs (and HMMs) in these tasks. In Subsection 5.2.3 we also briefly talk about their application to problems other than NMT. LSTMs were introduced by Hochreiter and Schmidhuber (1997), who also first discovered the vanishing gradient problem. GRUs have only been introduced recently by Cho et al. (2014b). Because both LSTMs and GRUs are based on a concept called gates, for simplicity we will henceforth refer to them as gated units when we are talking about either one of these two.

For a better understanding of how gated units work, imagine the neurons of the RNN to be a black box. Then, gated units work exactly like RNNs, i.e. they compute the hidden state s_t based on the previous hidden states and the input. In the network introduced in the last Section, the hidden states were calculated as $s_t = \tanh(Ux_t + Ws_{t-1})$. All these two new units do is compute s_t in a different way. Instead of just replacing all traditional units, they can even be combined such that a network has both, traditional and LSTM or GRU units.

Of course, the interesting part of these more advanced units is what is happening in these black boxes. In essence, gated units are smart in the sense that they can remember a value for a much longer amount of time – unlike common RNNs whose long-term memory deteriorates due to the vanishing gradient problem.

They are called gated because they feature gates that take input activations from the input at the current time step as well as from the hidden layer from the previous time step. This is usually accomplished by applying a sigmoidal function σ. If the gate's value is zero, the flow will be cut off. If it is one, all information is passed through.

First, let us take a closer look at LSTMs. These units contain three gates, namely:

- The forget gate f allows the network to learn to flush the contents of an internal state c_t, i.e. it decides when the network should forget.
- The input gate i determines if the input is significant enough to remember.
- The output gate o determines the strength of the output.

We are no experts on how the human brain works, but this does sound a lot like something we would do as we receive new inputs from our environment. Some things we forget instantly, other things appear to be significant enough to remember while others seem to require some intervention.

Above we have annotated these gates using the letters i, f, o and now we will define the equations for these gates illustrated in Figure 5.9. The value for the forget layer f is computed as

$$f_t = \sigma(x_t U^f + s_{t-1} W^f) \tag{5.35}$$

where σ is the logistic function as per Equation (5.3). The sigmoid function *squashes* the value of its argument between 0 and 1 through element-wise multiplication by looking at the previous state s_{t-1} and the input x_t at time t. A value of 1 means to certainly keep this memory while a value of 0 means to completely forget it.

The input gate i is defined by

$$i = \sigma(x_t U^i + s_{t-1} W^i) \tag{5.36}$$

and decides how much of the newly computed state given the current input contributes to the final result for this unit. The forget layer f is given by

$$f = \sigma(x_t U^f + s_{t-1} W^f) \tag{5.37}$$

and defines how much of the previous state is passed through. Alternatively, in other words, how much of it is to be forgotten. The output gate o is defined by

$$o = \sigma(x_t U^o + s_{t-1} W^o) \tag{5.38}$$

and decides what is going to be outputted. The candidate hidden state g is computed just as in our traditional RNN:

$$g = \tanh(x_t U^g + s_{t-1} W^g) \tag{5.39}$$

However, it is not taken directly into account. Instead, it is used in conjunction with the input gate:

$$c_t = c_{t-1} \circ f + g \circ i \tag{5.40}$$

(5.41)

where c_t is called the internal memory of the LSTM unit. The memory is based on the previous memory c_{t-1} and also takes into account the forget gate f, while the new candidate state g takes into account the output of the input gate i.

The final output of a node is then given by applying the tanh function, and then multiplying the result with o, i.e.

$$s_t = \tanh(c_t) \circ o. \tag{5.42}$$

In essence, these equations describe how values flow into and out of the memory block c_t and how much they contribute to the final state s_t.

As can be seen from Figure 5.9, there may be infinite ways to construct an LSTM. For example, a modern variant developed by Gers and Schmidhuber (2000) adds *peepholes connections* to the LSTM's multiplicative gates so that the gate layers σ take the cell states c_{t-1} and c_t into account, which allows them to learn precise timings.

The GRU on which our work is based is also a recent variant of an LSTM by Cho et al. (2014b). Chung et al. (2014) compared both, but no conclusion as to which one is better could be made. Greff et al. (2015) also compare different variants of LSTM architectures for RNNs. The researchers performed large-scale analysis of eight different variations and tuned the hyper-parameters using random search. They conclude that the vanilla LSTM architecture performs well and using any of the studied varieties does not significantly improve the model's performance. For GRUs this is, however, an interesting outcome, because they simplify the LSTM model by coupling the input and forget gate, skipping an internal memory signal, and do not have any peephole connections.

This brings us to the last part on the foundations of neural networks for NMT: The GRU. The GRU is a simplified version of an LSTM that combines the input gate i and the output gate o into a single update gate z:

$$z = \sigma(x_t U^z + s_{t-1} W^z) \tag{5.43}$$

This single gate decides how much of the previous memory to persist. A GRU also adds a reset gate r that combines the current hidden state and the previous hidden state

$$r = \sigma(x_t U^t + s_{t-1} W^r) \tag{5.44}$$

It also adds a gate z, which can be seen as a combination of the input and forget gate of an LSTM. This gate is defined by:

$$z = \sigma(x_t U^z + s_{t-1} W^z) \tag{5.45}$$

The output of this unit is given by

$$s_t = (1 - z) \circ \tilde{s}_t + z \circ s_{t-1} \tag{5.46}$$

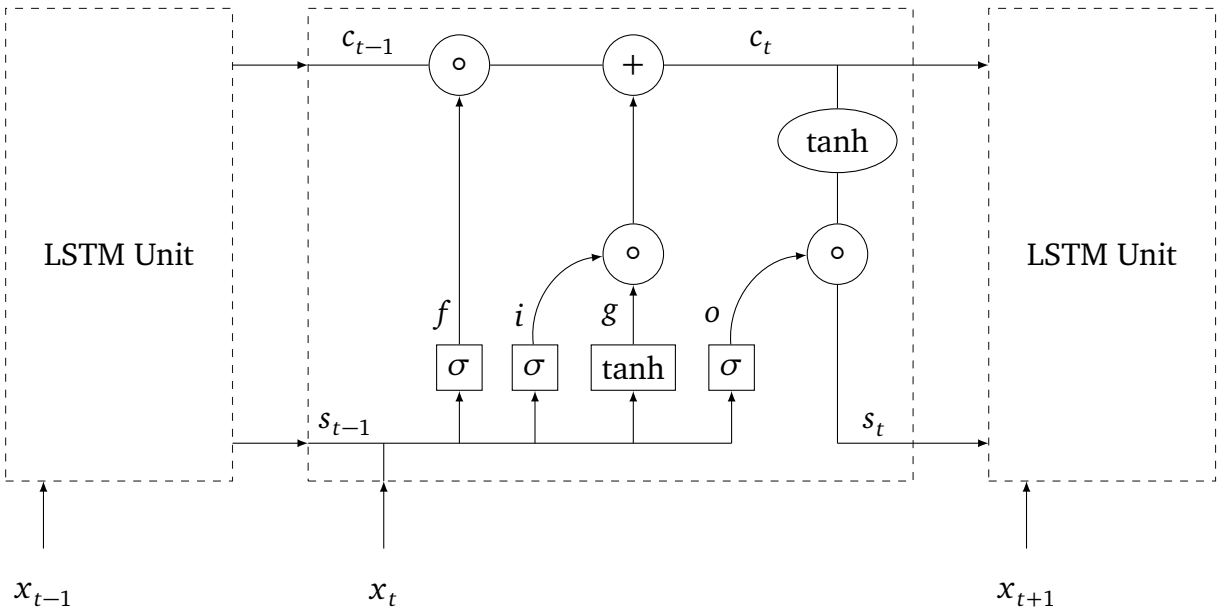

Figure 5.9: The flow of information in a LSTM unit. A LSTM features a special internal memory c_t that is carried along the time axis (top). The singals f, i, o are called the forget, input, and output gates that determine how much of the input should be forgotten, remembered or outputted, respectively. The candidate hidden state g is taken into accound when computing the output hidden state s_t based on the previous hidden state s_{t-1}.

where $\tilde{s}_t$ is defined as:

$$\tilde{s}_t = \tanh\left(x_t U^h + (s_{t-1} \cdot r) W^h\right). \tag{5.47}$$

Comparing LSTMs and GRUs, we see that both feature an additive component of their update from t to $t+1$. Hence, both are able to remember long-term dependencies and the vanishing gradient problem is experienced to a lesser extend. Chung et al. (2014) provides more literature on the differences and similarities between these two units.

Chung et al. (2015) extend LSTM/GRU-based RNNs even further culminating in an architecture called Gated Feedback Recurrent Neural Network (GR-RNN) by adding a mechanism that allows for information to flow from recurrent upper layers to lower recurrent layers using a global gating unit for each pair of layers.

5.2.3 Other Applications of RNNs

To provide a better overview of what RNNs are capable of, we briefly explore some applications other than NMT. Historically, RNNs were thought to be extremely hard to train. They have been around for decades but their full potential has only been recognized recently, and advances such as LSTM and GRUs coupled with a much better computation speed on graphics cards on top of freely available software libraries have made them very successful in many areas.

One such success can be found in the area of handwriting recognition where Alex Graves et al. (2009) achieved state-of-the-art results using LSTMs performing considerably better than the

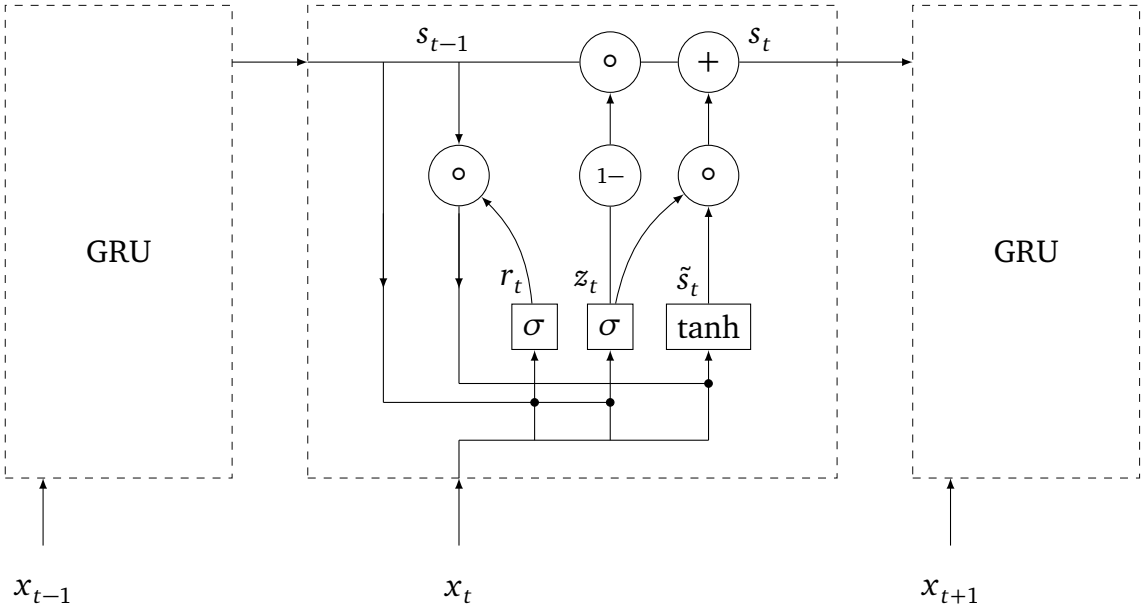

Figure 5.10: The flow of information in a GRU. Most notably and as opposed to a LSTM unit, the GRU does not feature an internal memory signal. The reset gate z_t can be seen as an alternative to the LSTM's memory because it decides how to combine the input with the previous state. The input gate i and the output gate o from a LSTM unit were further combined into a single update gate z_t.

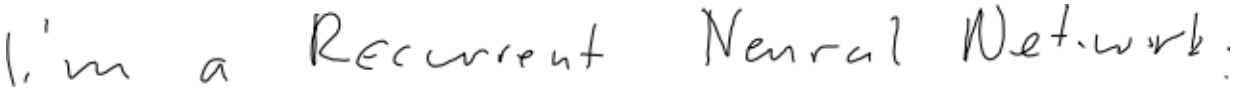

Figure 5.11: Handwriting Synthesis of an LSTM model by Alex Graves (2013).

previous HMM state-of-the-art. The same author has also done the opposite, i.e. he performs handwriting synthesis based on a text sequence. The output of this synthesis model whose input was *I'm a Recurrent Neural Network* is illustrated in Figure 5.11. To generate these real-valued sequences, the models were trained on handwriting recordings made up of as pen-tip locations.

LSTMs have also been applied to automatic speech recognition systems by Alan Graves, Mohamed, and Hinton (2013). Speech recognition is an especially hard task due to much variabilities such as accents, dialect, style, pronunciation, and noise. The interaction between these factors is complicated and highly nonlinear. It is important to understand that the application of RNNs to speech recognition and the related improvement in accuracy is not incremental, i.e. it is a fundamental change in acoustic modeling. Researches have spent over 30 years building algorithms like Dynamic Time Warping (DTW), engineering features such as the Mel-frequency cepstral coefficient (MFCC), adapting to speakers using Vocal Tract Length Normalization (VTLN) and so on. As it is often the case with neural networks, instead of designing highly complicated features we learn useful feature representations. Hence, many of these handcrafted methods are not required anymore with RNNs, leading to much simpler learning systems. Indeed, RNNs have brought a massive increase in accuracy over the traditional HMM models with highly complicated features.

Another interesting application with no particular purpose is shown by Andrej Karpathy in his blog post [1]. Nonetheless, it provides a detailed insight into how RNNs learn structure as the training continues. He trains RNNs on text data to generate new text character by character. Similar to character-based NMT, characters are encoded into a vector using 1-hot encoding and fed into an LSTM. Trained on Leo Tolstoy's War and Peace, the model generates samples every 100 iterations, and interestingly the progress it makes over time seems very natural. At first, the model generates random characters with no visible structure. However, at some point the model learns that words are separated by spaces. After some time, it also recognizes that quotes need to be balanced and that periods end a sentence. At last, it even manages to spell most words correctly. Of course, it does not produce good literature and produces mostly absurd sentences, but it gives us an idea of how these model learn.

5.3 Advantages over Hidden Markov Models

Hidden Markov Models (HMMs) were introduced in Section 3.1 using the toy example of Part-of-Speech tagging. In general, these models are often used for predicting sequences and allow us to infer unobservable hidden events based on observable events and can also be applied to machine translation. In this case, the words in the source language are the observable events while the words in the target language are the unobservable events.

However, there are many problems with HMMs. Their state must be drawn from a discrete state space S. The Viterbi algorithm (Viterbi 1967) used to find the most likely sequence of hidden state scales in time $O(|S|^2)$. In machine translation where we have hidden states in the range of tens of thousand, the standard operations become infeasible for Markov models.

In the previous section, we have also explored the vanishing gradient problem in RNNs and seen possible solutions in the form of LSTMs and a GRUs. We have talked about the importance of a model being able to capture long-term dependencies in case of language translation. In an HMM, each hidden state can only depend on the immediate previous hidden state, making the model unable to capture long-term dependencies. It is possible to extend an HMM for it to feature a larger context window; however the state space grows exponentially with the size of this window. Hence, Markov models are impractical for modeling long-range dependencies (Alex Graves, Wayne, and Danihelka 2014).

As we have shown in the previous section, in neural networks we can capture long-term dependencies by using LSTMs or variants.

5.4 Language Translation using Recurrent Neural Networks

Neural Machine Translation (NMT) is a recently proposed approach to machine translation based purely on neural networks. Neural networks were previously used to augment existing machine translation systems, e.g. for reranking the n-best list outputted by phrase-based machine translation. For an overview of phrase-based machine translation, please refer to Section 4.1. However, we will not concentrate on using NNs to improve phrase-based systems as the novelty in NMT is that neural networks are used at the core of the translation system.

[1] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Just like phrase-based translation models, NMT models are based on the statistical approach to machine translation, which is called SMT. In SMT, we propose multiple possible translations and choose the candidate with the highest probability and train our model using parallel text, i.e. text that is written in e.g. German and English simultaneously. When we want to translate from German to English, we call x the German source sequence and y the English target sequence. A SMT model's translation is not a deterministic function but rather a conditional probability of y given x, i.e. $p(y|x)$. Then, the goal of NMT is to maximize the conditional probability, i.e. $\arg\max_y p(y|x)$.

Given this notation, we can now define a score function that determines how well a possible translation explains the source sequence. This score function is known as the *log-likelihood* and is given by

$$\log p(y^{(n)}|x^{(n)}, \theta) \tag{5.48}$$

where θ are model-defining parameters and n is the index in our training data, i.e. $x^{(n)}$ corresponds to the n-th source sentence, while $y^{(y)}$ corresponds to n-th target sentence. Having more than one training example, we define our training set to be:

$$D = (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots (x^{(n)}, y^{(n)}) \tag{5.49}$$

The training set D is just a collection of source and target sequences. The overall likelihood or *score* of a model for a particular data set D is then simply the sum of the likelihoods of each source and target sequence pair:

$$L(\theta, D) = \sum_{(x^{(n)}, y^{(n)}) \in D} \log p(y^{(n)}|x^{(n)}, \theta) \tag{5.50}$$

The goal here is to find parameters θ that maximize this score. This concept is also known as the *maximum likelihood estimator* (MLE). Of course, the interesting question now is how exactly we model $\log p(y^{(n)}|x^{(n)}, \theta)$.

In Subsection 6.2.3 we encoded our source and target sequences into indices in an array of size V, which we called our vocabulary. Now we will denote the variable-length source sequence as $x = (x_1, \dots, x_{T_x})$ in which x_t are 1-of-V encoded vectors depending on the t-th word in the sequence. For example, given the sequence $(4, 40, 80)$ and a vocabulary with a size of 100, the resulting sequence contains 3 vectors where the first vector contains all zeroes except for the third entry and so on. The variable-length target sequence is analogously denoted as $y = (y_1, \dots, y_{T_y})$.

5.4.1 Encoder-Decoder Framework

Neural networks work well in many areas, but when it comes to sequence-to-sequence learning problems arise. Consider the German sentence $x = (\text{Wie}, \text{geht}, \text{es}, \text{dir}, ?)$ and its English translation $y = (\text{How}, \text{are}, \text{you}, ?)$. The German source sentence x is composed of 5 tokens while the

Table 5.1: Notation used in our introduction to NMT.

Notation	Description
T_x	Input sentence length
T_y	Target sentence length
$x = (x_1, \ldots, x_{T_x})$	Tokens (e.g. words) in the *source* sequence
$y = (y_1, \ldots, y_{T_y})$	Tokens (e.g. words) in the *target* sequence
t	Time step in either the decoder or encoder
i	Time step in the *decoder*
j	Time step in the *encoder*
h_j	Hidden state in the *encoder* RNN
z_j	Hidden state in the *decoder* RNN

English target sentence consists of 6 tokens. Vanilla neural network learning methods rely on the input and target to be encoded with vectors of fixed dimensionality, but for language translation, the length of x and y is not fixed but variable and not known apriori.

Several researchers (Sutskever, Vinyals, and Le 2014; Cho et al. 2014b; Bahdanau, Cho, and Bengio 2014) have suggested solutions that address the problem of mapping variable-length input sequences to variable-length output sequences. Most of them are based on an *encoder-decoder* architecture that maps the variable-length input sequence to a fixed-size vector using one RNN (the encoder), from which another RNN (the decoder) starts generating words in the target sequence until a special doken denoting the end of the sequence is generated.

Before we start describing these models, we want first to clarify the notation we use to provide a consistent notation when we present different architectures. Hence, our notation can be found in Table 5.1.

Using this notation, the encoder-decoder framework can be formalized as

$$h_t = f(x_t, h_{t-1}) \tag{5.51}$$

$$c = q(h_1, \ldots, h_{T_x}) \tag{5.52}$$

where h_t denotes the hidden state at time t based on the input x_t and the previous hidden state h_{t-1} using an activation function f (e.g. LSTM). The vector c is called the context vector and is generated from the sequence of hidden states h_t and some function q.

The joint probability of target words given source words can be defined as the product of the conditional probability of a word at time t given all previous words and the input context c as follows:

$$p(y_1, \ldots, y_{T_y} | x_1, \ldots, x_{T_x}) = \prod_{t=1}^{T_y} p(y_t | y_1, \ldots, y_{t-1}, c). \tag{5.53}$$

The conditional probability of y_t given the previous words and the context is defined as:

$$p(y_t | y_1, \ldots, y_{t-1}, c) = g(y_{t-1}, z_t, c) \tag{5.54}$$

where g is a function outputting the probability of y_t given the previous word $yt-1$, the hidden state z_t, and the context vector c.

The calculation of the context vector c shown in Equation (5.52) was purposely expressed in a more sophisticated manner because we are now going to describe different types of encoder-decoder frameworks that can be reduced to the equations above.

To be more specific, let us explain how to build an encoder-decoder architecture to address sequence-to-sequence learning problems in machine translation. The vectors s_j represent words in continuous-space and are projections of the 1-of-V dimensional vectors of the input words w_j using the weight matrix matrix $E \in \mathbb{R}^{M \times V}$ where V is the size of the vocabulary and M is the size of word embeddings. Formally, the projections can be computed by

$$s_j = Ew_j. \tag{5.55}$$

The vectors s_j are the word representation in continuous-space learned from data and were summarized through the use of an RNN, i.e.

$$h_j = f(h_{j-1}, s_j). \tag{5.56}$$

This concept is illustrated in Figure 5.12 (left). Given the German input sentence, the first step is to convert word indices in the source sentence to 1-of-V vectors w_j. Then, these vectors are projected to s_j using the embedding matrix E. The RNN is used to summarize the continuous-space vectors s_j and this culminates in the last hidden state h_{T_x} with $T_x = 7$. This vector can be seen as an intermediate format or summary that hopefully captures all information in the input sequence required to propose a translation. Not shown in this figure is the End-of-Sequence (EOS) marker w_8 that is inserted after the last token in the input sequence and marks the end of the sentence.

Sutskever, Vinyals, and Le (2014) show an interesting property of an RNN encoder that summary vectors h_{T_x} of sentences having similar meaning are close to each other in 2-dimensional space. In other words, it learns not only syntactic structures of sentences, but preserves also the meaning of sentences. For example, the sentence *Mary admires John* is very close to *Mary is in love with John*, while *John respects Mary* is further away.

The decoder shown on the right in Figure 5.12 is essentially the encoder in reverse. The RNN's internal state z_i is calculated based on the summary vector h_{T_x} outputted by the encoder, the previous word u_{i-1} and the previous hidden state of the decoder z_{i-1}. Formally, the hidden states in the decoder are computed by:

$$z_i = f\left(h_{T_x}, u_{i-1}, z_{i-1}\right) \tag{5.57}$$

Each word k in the target vocabulary can now be scored based on the probability that it comes after the previous predicted word given the source sequence. This probability is denoted as p_i in

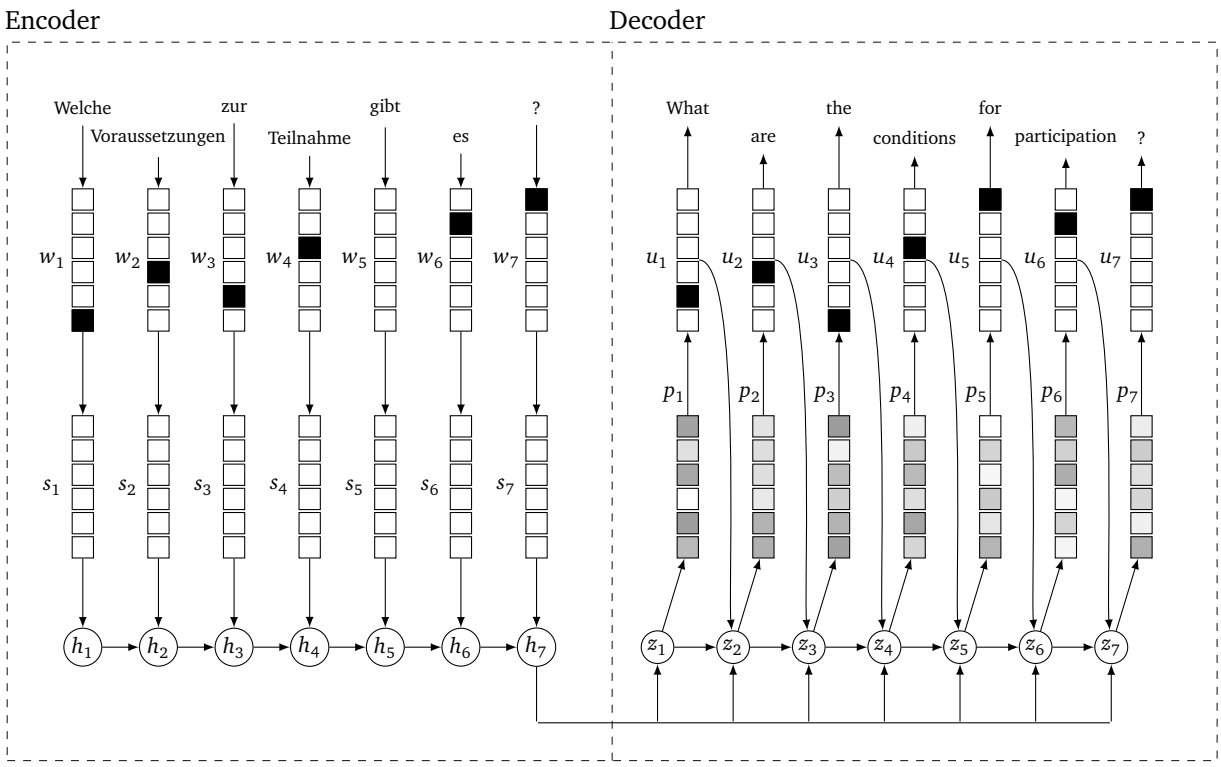

Figure 5.12: NMT encoder (left) and decoder (right). The 1-of-V encoded words $w_1, \ldots, w_7$ are projected to vectors $s_1, \ldots, s_7$ in continuous-space and compressed into the summary vector h_7 using an RNN. This summary vector of the input sequence is fed into the hidden nodes $z_1, \ldots, z_7$ of the RNN in the decoder. Based on these hidden nodes, probabilities for target words $p_t, \ldots, p_7$ are computed, which are then turned into 1-of-V encoded word vectors $u_1, \ldots, u_7$ again until a special End-of-Sequence (EOS) marker is found at time T_x. The EOS markers are omitted in this figure, but in the input sequence it would be found at time $j = 8$ and the model is likely to generate it at time $i = 8$ in the decoder as well.

Figure 5.12. Given the summary vector h, the target word u, the hidden state z, and the bias b, we can define the unnormalized score $e(k)$ for a word k in the target vocabulary as:

$$e(k) = U^s(h + u + z) + b \tag{5.58}$$

These scores can now be turned in probabilities using softmax:

$$p(u_j = k | u_1, \ldots, u_{j-1}, h_{T_y}) = \frac{\exp(e(k))}{\sum_j \exp(e(j))} \tag{5.59}$$

Using the probability distribution over the target words, we can now produce a translation by selecting words repeatedly until we hit an EOS marker. In Figure 5.12 the EOS markers are omitted, but the model is highly likely to generate it at the end of a translation sentence due to learning this from the training data, where it is inserted manually.

Using the parallel training corpus D defined above, we can train our models by maximizing the log-likelihood. For any instance, we can compute the conditional log-probability $\log(y^{(n)})|x^{(n)}, \theta)$ and the log-probability is hence given by

$$L(D, \theta) = \frac{1}{N} \sum_{n=1}^{N} \log p(y^{(n)}|x^{(n)}, \theta) \tag{5.60}$$

As described in Section 5.2, the log-likelihood function can be optimized using stochastic gradient descent and BPTT.

5.4.2 Soft Attention Mechanism

One major limitation of the encoder-decoder architecture discussed in the previous section is that the translation quality according to BLEU (see Subsection 4.2.1) deteriorates as the length of the source sequence increases given a small encoder-decoder model (Cho et al. 2014a).

This problem is due to the input sequence being compressed into a fixed-length vector. Given that an RNN with a GRU still has problems remembering long-term dependencies, this leads to our model having forgotten how the sentence started by the time we are at the end of the sentence.

To alleviate this problem, Bahdanau, Cho, and Bengio (2014) extend the vanilla encoder-decoder architecture by allowing it to automatically search for relevant parts in the source sentence that are relevant for predicting a target word. This component is called *soft attention* and improves the performance of NMT to a point where it reaches state-of-the-art phrase-based SMT. The soft attention mechanism is realized by including a small FNN in the decoder. Another interesting property of the attention mechanism is that it produces word alignments without supervision, which we will later exploit to deal with words missing from the vocabulary.

This architecture also contains a Bidirectional Recurrent Neural Network (BRNN), which we have briefly described in Section 5.2. A BRNN is a combination of two RNNs:

- The forward RNN $\overrightarrow{f}$ reads the input sequence from x_1 to x_{T_x}, i.e. it reads it from the left to the right. The hidden states from the resulting RNN are called *forward hidden states* $\overrightarrow{s_1}, \ldots, \overrightarrow{s_{T_x}}$.
- The backward RNN $\overleftarrow{f}$ reads the input sequence from x_{T_x} to x_1, i.e. from the right to the left. The *backward hidden states* are named $\overleftarrow{s_1}, \ldots, \overleftarrow{s_{T_x}}$.

Using this notation, the forward states $\overleftarrow{h_j}$ and the backwards states $\overrightarrow{h_j}$ can be concatenated, i.e.:

$$h_j = \left[\overrightarrow{h_j^T}; \overleftarrow{h_j^T}\right]^\top \tag{5.61}$$

We have also talked about RNNs (even using LSTM units) being able to memorize better more recent events. For an RNN this means all words preceding the current word x_j, and for a BRNN this means words *around* x_j. The so-called *annotation vector* h_j can be seen as a good representation of the current word x_j. The use of this BRNN is depicted in Figure 5.13 in which

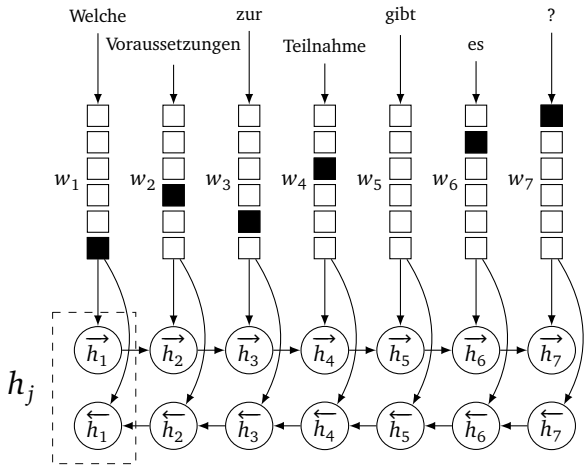

Figure 5.13: Encoder using a Bidirectional Recurrent Neural Network with forward states and backward states. The *annotation* vector h_j is the concatenation of both the forward state and backward state at time j.

the new combined hidden state at time t would be the concatenation of both the forward state at time t and the backward state at time t.

This attention-based model, which is depicted in Figure 5.12, redefines the conditional probability from Equation (5.54) as:

$$p(y_i|y_1,\ldots,y_{i-1},x) = g(y_{i-1},z_i,c_i) \tag{5.62}$$

where z_i is an RNN hidden state in the decoder at time i given by

$$z_i = f(z_{i-1},y_{i-1},c_i) \tag{5.63}$$

where c_i are *distinct* context vectors depending on the annotation vectors $(h_1,\ldots,h_{T_x})$. The context vectors c_i are defined as

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{5.64}$$

where h_j are hidden states in the encoder and α is a weight matrix given by the softmax function

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \tag{5.65}$$

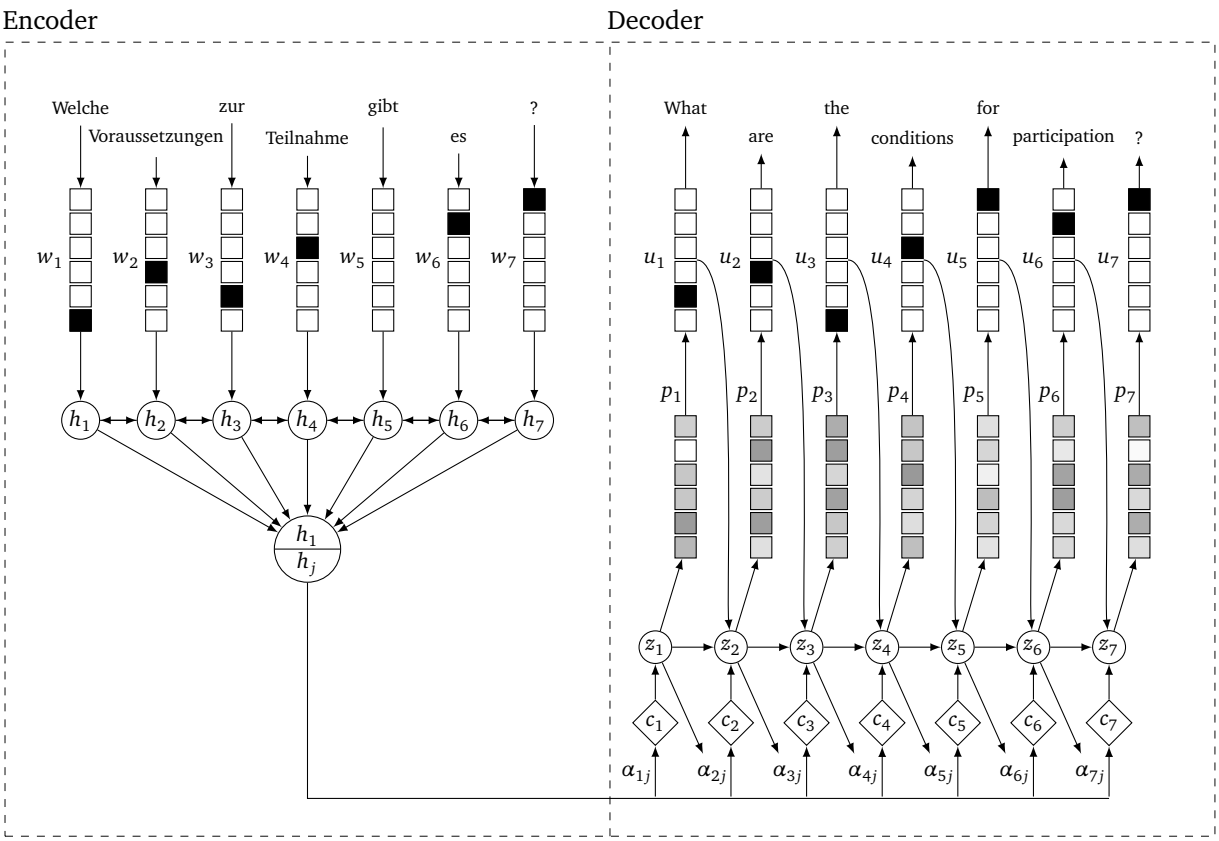

Figure 5.14: NMT with attention mechanism and a bidirectional RNN in the encoder. The distinct context vectors c_i are defined for every word in the target sequence. Given a target word at position i, the mechanism is able to give attention to a source word at position j by weighting the context vector c_i using the weights α_{ij} (softmax).

which assigns a weight to each annotation h_j. The so-called *(soft) alignment model*

$$e_{ij} = a(z_{i-1}, h_j) \tag{5.66}$$

aligns the input token at time j and the output token at time i. The model a is implemented as a small FNN with a single hidden layer and is called the *attention mechanism*. The scalar e_{ij} is called energy and associates a relevance score for every word in the source sequence with every word in the target sequence.

The *softmax* normalization (Bridle 1990) performed in Equation (5.65) guarantees that all scores for a source word sum up to 1 and this helps us interpret it as posterior probabilities. Softmax derives from log-linear models and allows us to explain the alignment in terms of odds ratios.

Because the context vectors c_i in Equation (5.64) are computed based on the annotation vectors h_j and their corresponding weights, it allows the model to assign special importance to a particular word in the source sequence.

There are two types of attention mechanisms: *hard* attention is a stochastic variant trainable by maximizing a variational lower bound. On the other hand, *soft* attention makes the objective function smooth and differentiable, so that it can be traine via standard backpropagation.

The alignment weights in the matrix α_{ij} can be visualized, and this has been done in Figure 5.15. Note that we clipped weights below 0.1 in this illustration, the color refers to the strength of the

alignment weight. By looking at various examples, we noticed that for nouns, the alignments are mostly very strong and this is intuitive since this is often a one-to-one relationship.

The architecture by Sutskever, Vinyals, and Le (2014) is possibly the most intuitive and straightforward architecture. It features a fixed-size context vector, i.e. due to the context vector c being equal to the final hidden state h_{T_x}, the size of this vector is known beforehand.

In principle, the encoder-decoder architecture in Cho et al. (2014b) and Sutskever, Vinyals, and Le (2014) is almost the same except for few minor changes such as reversing the input sentence, the number of recurrent hidden layers, and the unit type in the hidden layer. One important discovery made by Sutskever, Vinyals, and Le (2014) is that reversing the order of the input sequence improves translation performance for French to English by 4.7 BLEU points. The authors believe this is due the distance between a word in the source source sentence and its corresponding words in the target sentence. While the average distance is unchanged, by reversing words in the source sequence, the first few words in the source sentence are closer to the first few words in the target sequence.

The NMT design by Bahdanau, Cho, and Bengio (2014) differs in that it features an attention-mechanism and a BRNN in the encoder. We believe deploying a BRNN in the encoder makes reversing the input sequence unnecessary experiment because in essence the backward RNN already does so.

Generalizing this idea of an attention mechanism, we may refer to it as a neural network with memory. It has memory in the sense that by the time the decoder tries to predict a word, it remembers which words in the source sentence are important. In this case, the memory is restricted to a single example and does not extend beyond sentence boundaries. Sukhbaatar, Weston, and Fergus (2015) introduce a neural network with external global memory and apply it to tasks other than NMT. Closely related are *Neural Turing Machines* by Alex Graves, Wayne, and Danihelka (2014) which are analogous to Turing Machines but differentiable end-to-end; hence it can be trained with gradient descent. Instead of considering all elements in the source sequence, Luong, Pham, and Manning (2015) have tried only to look at a subset of source words at a time. This may help when translating longer sequences such as paragraphs as opposed to sentences because the computation is less expensive. Their approach can be seen as somewhere between hard and soft alignment, i.e. words are neither aligned with a single word nor to all words in the sequence.

5.4.3 Multi-pass Decoding and the Out-of-Vocabulary Problem

Multi-pass decoding generally describes the concept of inserting additional passes into the MT pipeline after the decoder and is analogously used in ASR. The goal of multi-pass decoding is to replace unknown unknown words in the translated sentence and to rerank translation hypotheses.

A solution for the first problem is necessary because NMT uses a fixed-size vocabulary usually built from the k most frequenst words, thus making it is prone to OOV problems in which an unknown source word leads to an unknown target word. Hence, our first goal in multi-pass decoding is to recover from this problem by using a multitude of strategies, including copying the word from the source sequence and looking them up in a dictionary. The aligned words

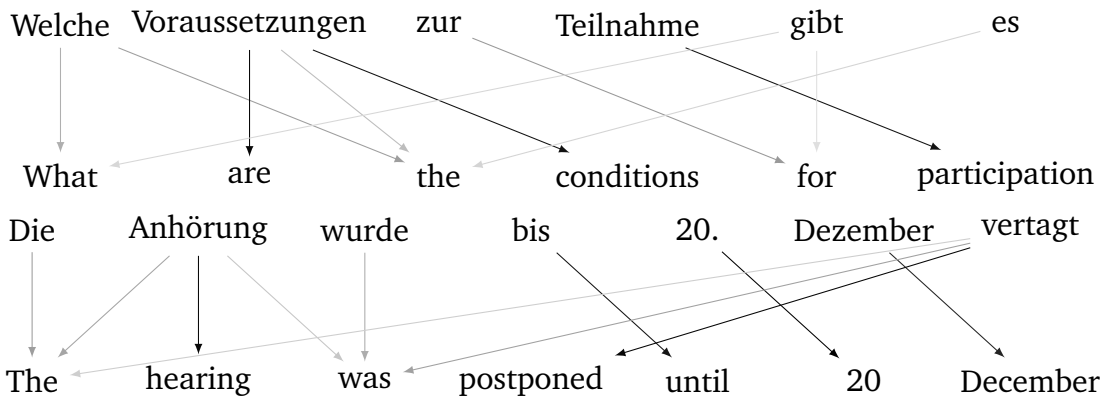

Figure 5.15: Weights of the alignment model for two exemplary sentences when translating a sentence from German to English. The color of the arrows corresponds to the alignment weight so that larger alignments feature a darker arrow.

in the source sequence for a target word can be found by querying the alignment weights in the attention mechanism of the NMT decoder. This concept is illustrated in Figure 5.16 where the word *Gutach*, which neither present in the source vocabulary nor in the target vocabulary, gets copied to the target sentence by making use of the alignment weights. Copying words also works for models that do not provide alignment information itself by using an external word aligner and this has been done by Luong et al. (2015). Instead of copying the word from the source sequence, we can also look it up in a dictionary. We will explore this dictionary-based replacement method later in our work and a similar technique has also been tried by Luong et al. (2015) at the same time.

Before searching for solutions to the OOV problem, it is worthwhile to further analyze the circumstances surrounding it. Given that the size of the vocabulary is arbitrary but fixed, it can only hold a finite set of words. We analyzed this problem by looking at the POS tags of unknown words. We first built a list of the $n = 30000$ most frequent words on a large text corpus, and then consulted another small corpus and counted the number of tags for words found in the vocabulary, normalized by the total number of appearance of each tag:

$$\text{coverage}(\text{tag}) = \frac{\#\text{ of words with } \textit{tag} \text{ in vocabulary}}{\#\text{ of words with } \textit{tag}} \tag{5.67}$$

This alone is not meaningful, because some tags may only appear at a very small ratio, making special treatment unnecessary. Hence, we also calculated how often a tag appears in relation to the total number of tags in the corpus:

$$\text{ratio}(\text{tag}) = \frac{\#\text{ of words with } \textit{tag}}{\#\text{ of words}} \tag{5.68}$$

The results of this study with $n = 30000$ can be found in Table 5.2. In this table, we reduced the different tagsets for German and English to common tags in order for them to be comparable. In

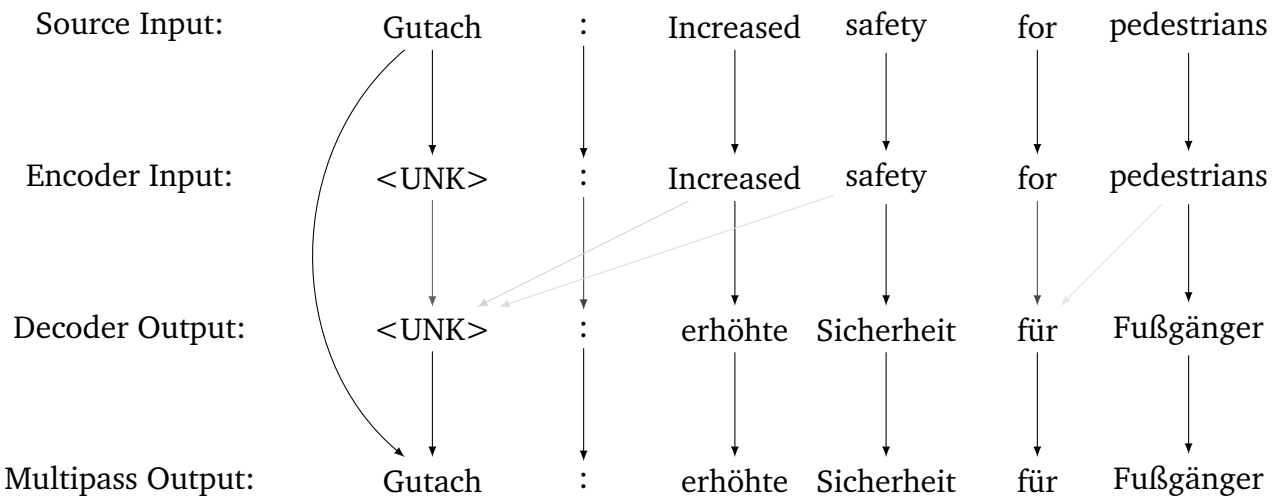

Figure 5.16: Multipass decoding where the source word *Gutach* is copied to the translation output by querying the alignment model.

particular, we combined all different types of adjectives into a single *ADJ* tag. The ratio refers to the frequency in the test set, e.g. 20.50% for *NN* means that 20.50% of the words in unseen data are nouns.

Our findings show that NEs in all forms are poorly covered, and this is intuitive given that a vocabulary cannot possibly contain the names of all people, places, or organizations. NEs also appear quite frequent, i.e. 5% of all unknown words are NEs.

The coverage of numbers is also suboptimal, with the English vocabulary covering 92.01% while the German vocabulary is covering only 87.62%. For numbers as well as for NEs this may not be a problem per se because OOV words can be copied from the source sentence using the alignment model, but it does raise the question whether these tokens should even be added to the vocabulary. If we were to skip out on these tokens when building the vocabulary, it might free up more space for words that cannot simply be copied from the source sentence. Hence, later in our work, we trained and evaluated such models.

The largest deviance for a POS tag in terms of coverage can be seen for nouns. English features a coverage of 93.25% while German covers only 68.73%. The reason for this is most likely due to an interesting property of the German language: compound words. In German, new words can be created by concatenating two or more words, which is also known as compounding and described in Subsection 6.3.2

Table 5.2: Coverage of words in unseen data by POS/NE tag.

POS/NE	Description	German		English	
		Coverage	Ratio	Coverage	Ratio
TRUNC	First word of German word composition, e.g. *An- und Abreise*	36.84%	0.12%		
I-PER	Named Entities (Persons)	37.03%	2.21%	41.69%	2.43%
I-ORG	Named Entities (Organizations)	51.67%	1.33%	76.70%	2.14%
I-LOC	Named Entities (Locations)	63.17%	1.30%	76.18%	1.56%
ITJ/UH	Interjection, e.g. *uh*, *ah*	66.67%	0.01%	70.00%	0.01%
NN	Nouns	68.73%	20.50%	93.25%	14.75%
VVIZU	Infinitive with the infix *zu*, e.g. *anzukommen*	70.37%	0.17%		
FM/FW	Foreign language word	70.76%	0.27%	61.90%	0.03%
ADJ	All forms of adjectives	78.99%	7.43%	90.49%	6.89%
VVFIN	Finite verb, e.g. *gehst* or *kommen*	86.08%	4.53%		
VVPP	Participle perfect, e.g. *gegangen*	87.13%	2.29%		
CARD/CD	Numbers, e.g. *4* or *42*	87.62%	1.65%	92.01%	1.85%
VVINF	Infinitive, e.g. *gehen*	90.11%	1.80%		

6 Experimental Setup

This Chapter describes the foundations of our experiments and specifically addresses preprocessing, which is an important part of the machine learning pipeline. Preprocessing is the process of transforming raw data into a format that is better suited as input to learning algorithms. Even though we are using RNNs that may require fewer preprocessing steps as opposed to the traditional machine learning methods, we put great emphasis on this pipeline step. In phrase-based SMT, preprocessing is known to improve the translation quality, yet one of the main selling points of NMT is that it does not require extensive preprocessing. Hence, our desire is to test this hypothesis. In this section, we provide a thorough overview of the preprocessing steps we utilized in our end-to-end translation system.

In the next Section, we describe the data sets utilized in our studies. Then, in Section 6.2, we describe the language-independent methods which are also independent as far as either language being on the source or target side is concerned. Finally, Section 6.3 deals with transformations that are specific to the German language.

6.1 Dataset

Our models were trained on the data provided by the 2014 Workshop on Machine Translation[1] (WMT). Specifically, we used the Europarl v7, Common Crawl, and News Commentary corpora. Our development set corresponds to the data in the *newstest2013* files while the test set is given by *newstest2014*. Unfortunately, there exist two types of test sets: one was released during the contest and is missing about 10% of the German/English data while another one released after the contest reinstates this missing data. We used the second one in our experiments, and often it is not clear which data set other NMT researchers used for papers released after the contest.

Table 6.1 provides statistics on the data used in our experiments. As can be seen from the table, we train our models on roughly 4 million instances and evaluate them on data sets containing only 3000 instances. It is important to note that while this may seem small in contrast to the size of the training data, it should be said that evaluation is very expensive in terms of computing power. Translating 3000 instances takes just under 24 hours on our computing cluster using 16 CPUs. Since we chose to run many evaluations as the training progresses, using more instances for evaluation becomes impractical.

[1] `http://statmt.org/wmt14/translation-task.html`

Table 6.1: Statistics for the training, development, and training corpora.

	Train		Dev		Test	
	English	German	English	German	English	German
Size in MB (uncompress)	277	310	0.36	0.38	0.36	0.4
Sentences	3839970	3839970	3000	3000	3003	3003
Words	89406382	84752166	64767	63354	67612	63078

6.2 Preprocessing

In this Section, we discuss all preprocessing steps which are shared by German and English. Subsection 6.2.1 describes a way of transforming raw input data into a sequence of tokens, i.e. into a sequence of words. In Subsection 6.2.2 we develop a filtering algorithm that removes training examples that are written in a language different from what is expected. Finally, we describe how a sequence of tokens is binarized such that it can be used by a learning algorithm in Subsection 6.2.3.

6.2.1 Tokenization

A tokenizer or segmenter describes a tool which performs lexical analysis, i.e. it converts a sentence into a sequence of tokens. The simplest form of a tokenizer divides a sentence using whitespaces. While this works for some cases, for a machine translation system it is usually insufficient. As it is always the case with natural language, there are many ambiguities involved, including:

- The period may be the final sentence symbol, part of an abbreviation (`U.S.A.`), part of a number (`1.200`), etc.
- The whitespace or comma character may be part of a number, i.e. `1 234`
- Single quotes are used to mark contractions and elisions, e.g. `don't`

However, some of these problems do not affect translation systems much. For instance, when a number or word is represented as two tokens instead of one, the system is very likely able to translate it correctly.

In our system, we use the tokenizer from Moses (Koehn et al. 2007), a Perl script[2] with regular expressions and a database of so-called non-breaking prefix. These prefixes are particularly important because sentences such as

```
Dr. John held a talk on neural networks.
```

would result in two sequences if the tokenizer did not take `Dr.` as non-breaking prefix into account.

[2] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/tokenizer.perl

Table 6.2: Effects of filtering by language detection. The common crawl corpus is the noisiest corpus and was especially affected by our filtering technique.

Corpus	# of instances (before filtering)	# of instances (after filtering)	change
Europarl	1920209	1828572	−4.77%
Common Crawl	2399123	2127071	−11.34 %
News Commentary	201288	194260	−3.49%
Total	4520620	4149903	−8.20%

6.2.2 Noise Reduction by Filtering

Bilingual corpora are often created using webcrawlers and may contain examples that are not written in the language they are supposed to be. For example, a German training corpus may contain sentences or sentence fragments in English, French, or Russian. As we do want our models to translate only to one language, we take steps to remove unwanted language material in our training data.

Our approach to filtering works as follows: We iterate over the source sentence sen_s and target sentence sen_t simultaneously and remove any $(\text{sen}_s, \text{sen}_t)$ pair if and only if the probability of either sentence being in the correct language is greater than 0.9, i.e.

$$P(\text{sen}_s \text{ written in } \text{lang}_s) \geq 0.9 \wedge P(\text{sen}_t \text{ written in } \text{lang}_t) \geq 0.9 \tag{6.1}$$

To detect the language the sentences are written in, we use the language detection toolkit `langdetect`[3]. This tool calculates the language probabilities from features of spelling using Naive Bayes with character n-grams.

The results of these filtering steps can be inspected in Table 6.2. The largest change (−11.34%) concerning instances filtered is present for the Common Crawl corpus. As this corpus is produced by a web crawler, it is also the one that contains the most noise regarding incorrectly classified languages at the time of creation. The other corpora are of higher quality and amount to a much smaller change.

6.2.3 Binarization

To be suitable for input into an NMT model, the input sequences (sentences) need to be converted from words to indices in an array. This process is called binarization and is performed after tokenization. To produce vector representations of sequences, we build a vocabulary of a fixed size that that contains the n most common words. This number corresponds to the vocabulary size in the model, and any word not found in the vocabulary is mapped to the special token `<UNK>`, which itself has an index in the vocabulary. A special end-of-sequence marker `</s>` is also

[3] `https://github.com/shuyo/language-detection`

used to denote the end of a sequence which mostly but not necessarily precedes punctuation. This marker is required because the length of translated sentences is not given at test time. When the model produces this marker, we are able to stop the decoding process when we encounter </s>.

A single sequence is binarized as follows:

$$\begin{array}{cccccccccccc} \text{We} & \text{believe} & \text{that} & \text{the} & \text{Council} & \text{'s} & \text{text} & \text{is} & \text{a} & \text{perspicuous} & \text{improvement} & . \\ \begin{pmatrix} 46 & 196 & 11 & 2 & 115 & 30 & 553 & 10 & 9 & 1 & 1800 & 4 \end{pmatrix} \end{array}$$

Note that the word *perspicuous* is not contained in the vocabulary and hence gets represented as 1, which maps to the <UNK> marker. Now we consider a second sentence:

$$\begin{array}{cccc} \text{Life} & \text{is} & \text{short} & . \\ \begin{pmatrix} 4285 & 10 & 547 & 4 \end{pmatrix} \end{array}$$

For batch training, given a batch of two sentences, these sequences are grouped together to form

$$\begin{pmatrix} 46 & 196 & 11 & 2 & 115 & 30 & 553 & 10 & 9 & 1 & 1800 & 4 & 30001 \\ 4285 & 10 & 547 & 4 & 30001 & 30001 & 30001 & 30001 & 30001 & 30001 & 30001 & 30001 & 30001 \end{pmatrix}$$

where 30001 refers to the index of the end-of-sequence marker. Also required for batch training is a mask that indicates which indices map to an actual word and not to the padding introduced above, i.e. for the example above the mask would be:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The reason to prefer batch training over training single instances is that it can be run much faster, especially on GPUs. Training is already almost exclusively done on GPUs, and a modern GPU is about five times faster than a modern CPU. In our implementation, binarization is done on the fly as opposed to storing a binarized version beforehand.

6.3 German Corpus Preprocessing

The German language is in many ways different to languages such as French and English, hence, we choose to do special preprocessing steps for German not required for other languages. In Subsection 6.3.1 we convert part of the training data from WMT14 to the new German orthography, and in Subsection 6.3.2 we split German compound words into their compounds for German being the source as well as the target language.

Algorithm 1: Filter old German orthography sentences

```
Data: corpus: Unfiltered German Corpus
Function filter(corpus, window_size = 3000, cutoff = 0.1):
    buffer ⟵ [];
    i ⟵ 0;
    count_new, count_new ⟵ 0;
    for line in corpus do
        i ⟵ i + 1;
        buffer ⟵ buffer + line;
        count_old ⟵ count_old + occurrences(daß, line);
        count_old ⟵ count_old + occurrences(muß, line);
        count_new ⟵ count_old + occurrences(dass, line);
        count_new ⟵ count_old + occurrences(muss, line);
        if i % window_size == 0 then
            ratio ⟵ (count_old + 1)/(count_new + 1);
            if ratio < cutoff then
                output buffer
            buffer ⟵ [];
```

6.3.1 German Spelling Conversion

Due to a portion of the train set being written before the German orthography reform of 1996, we designed an algorithm to convert this portion of the train set to the orthography in place after 1996. This is necessary because when translating from English into German, we do not want our system to translate to words written in the old spelling as this would impose a penalty concerning machine translation evaluation scores due to the words written differently. For the other way around, i.e. German being the source language, this would not be a problem and simple rules such as ß → ss would suffice. This rule is not always correct according to the terms of the German orthography, yet this would have solved the problem by creating one dictionary entry for the same word in case it is written in two different forms.

In either case, having multiple spellings of words results in these words being present in our limited dictionary twice, hence reducing the same word to exactly one dictionary entry is appropriate. For practical reasons, we use an algorithm to convert the spelling for German being on the target side as well as for being the source language. We first build a frequency distribution over the words of a filtered data set that only contains German words written according to the new orthography rules. Since the corpus is ordered chronologically, it is very likely that there exists a clear cutoff point where the majority of words is written in the new style.

For filtering, we simply introduce a sliding window of size 3000 and count the occurrences of *dass* as opposed to *daß* as well as for *muss* and *muß*. If the ratio of these counts is in favor of the new spelling (*dass* and *muss*), the sentences in this window is included, otherwise excluded (Algorithm 1).

We then build a frequency distribution D over the words of the filtered corpus, i.e. we count the words using a dictionary. This distribution is employed used in a second algorithm, which works by applying a set of regular expression rules E resulting in a list of candidates for each word. The most frequent of these candidates is then used as a substitution of the original word. Do note that the original word itself is part of the candidate list, so that this word is used in case it is the most frequent one according to D. The regular expression set E is composed of the following rules:

- (.*)ß(.*) ⟶ \1ss\2 (e.g. muß ⟶ muss)
- (.*)Ph(.*) ⟶ \1F\2 (e.g. Phantasie ⟶ Fantasie)
- (.*)([a-z])\2(.*) ⟶ \1\2\2\2\3 .(e.g. Bestelliste ⟶ Bestellliste)
- (.*)iell$ ⟶ \1ziell (e.g. essentiell ⟶ essenziell)
- (.*)ieen$ ⟶ \1ien (e.g. geschrieen ⟶ geschrien)

The development and test sets are not affected by old orthography; hence we did not apply any preprocessing.

6.3.2 Compound Words and Compound Splitting

Compounds are lexemes (less precisely, words) consisting of more than one stem. Compounding is found in many languages, including Dutch, Finnish, and German. To build a compound word, two or more words are joined resulting in a longer word. These words are problematic for MT when using a fixed-sized vocabulary such as in NMT because the dictionary needs to be large enough to accommodate every single word in a language. The existence of compound words in German means that the dictionary size needs to much greater as opposed to languages that do not contain many compound words like English. For example, the compound word *Stacheldraht* is produced by concatenating the words *Stachel* (barb) and *Draht* (wire), yet in German, it needs to be encoded in the dictionary explicitly while the English equivalent *barbed wire* is likely to be already encoded due to encountering other sentences.

As Germanic languages such as German have a high number of compound words, we explore the methodology of compound splitting for German being both on the source and target side. Usually, there are no isomorphic equivalents to compound words in most other languages, including English, and they rather correspond to multi-word terms. Compound splitting brings German into this multi-word form as well; hence an improvement in translation quality is very likely.

Two main techniques are often employed in compound splitters (Fritzinger and Fraser 2010; Popović, Stein, and Ney 2006):

- Corpus-driven approach: Based on word frequencies built on a training corpus, the best split is selected among the candidates.
- Linguistics-motivated approach: Words are split through linguistic analysis (e.g. by using a morphological analyzer).

It should be said that hybrid approaches are possible, and corpus-driven splitters found in the wild often contain some additional rules instead of relying purely on corpus statistics. Likewise,

while it is possible to split words based solely on the output of a morphological analyzer, a splitter may take word frequencies into account.

We studied the application of two splitters from both of these categories in our work:

1. The compound splitter developed by Weller and Heid (2012), herein referred to as *IMS splitter*, which is based on the corpus-driven approach by Koehn and Knight (2003). We compute a frequency list of lemmatized words using TreeTagger (Schmid 1994; Schmid 1995) on the WMT14 training data as well as a frequency list of words with respect to their POS tag. We switched to TreeTagger from the Stanford Tagger due to the much larger processing speed of the former. These two lists are used to improve the performance of the IMS splitter, and different splitting possibilities are ranked by the geometric mean of the frequencies. The score is given by

$$\text{score} = \left(\prod_{i=1}^{n} \text{freq}(p_i) \right)^{\frac{1}{n}} \tag{6.2}$$

   where p_i are the parts of the respective splittings and n the number of parts. An exemplary output of this splitter can be found in Table 6.3, which shows different ways to split a word along with the scores for this particular split. Because the IMS splitter only outputs lowercase by default, we uppercase words that are thought to be nouns according to the splitter. This splitter is does not solely rely on corpus statistics and additionally contains some handcrafted rules specific to the German language.

2. Fritzinger and Fraser (2010) studied compound splitters in terms of their effectiveness in SMT systems and propose a hybrid approach that uses corpus statistics as well as a morphological analyzer. Hence, we will refer to this splitter as *hybrid splitter*. Based on split points provided by the morphological analyzer *Smor*, word frequencies from a large training corpus are consulted. Smor is a finite-state based morphological analyzer concerned with the word formation of German, i.e. inflection, derivation, and compounding. By employing a hybrid approach, only linguistically motivated splittings (provided by Smor) are performed according to the frequency lists on the training corpus, hence reducing the number of incorrect splits. For example, The word *Durchschnittauto* (*average car*) is split into *Durchschnitt Auto* instead of *Durch Schnitt Auto*. The word *Durchschnitt* has a one-to-one relation with the English word *average*, but is itself a compound word.

Another important aspect when talking about compound splitting is the concept of *filler letters*. When a new word is formed through the use of compounding, sometimes filler letters are inserted between two compounds. For German, these filler letters are usually *s, en, n, er, and es*. For example, the German word *Jahresverpflichtung* is made from the words *Jahr* (year) and *Verpflichtung* (commitment), and in between these two words the infix *es* is inserted. The easiest way to deal with this issue is to use a list of filler letters and derive a rule in order tests if a compound ends with such a filler letters. Then, these letters can be removed, leading to proper compounds, i.e. words that exist. Indeed, the IMS splitter uses exactly these rules to provide proper splits. The hybrid splitter does not need an explicit definition of filler letters as this knowledge is already encoded in Smor.

Table 6.3: Exemplary output of the IMS splitter.

Compound	Split Compound	Score
kühleinrichtung	kühl_ADJ einrichtung_NN	869.3
	kühlen_V einrichtung_NN	251.4
	kühleinrichtung_NN	6.0
gezeitenkraftwerk	gezeiten_NN kraft_NN werk_NN	984.9
	gezeiten_NN kraftwerk_NN	324.44
	gezeitenkraft_NN werk_NN	243.6
	gezeitenkraftwerk_NN	33.0

7 Experimental Results

In this chapter, we present the results for the experiments we conducted, and explain the way we trained our models. In previous chapters, we have explained the inner workings of NMT in Chapter 5 as well as general preprocessing procedures such as tokenization and filtering in Chapter 6.

This chapter is divided into two major sections containing experimental results: Section 7.3 focuses on a variety of preprocessing steps, including lowercasing in Subsection 7.3.1, replacing NEs in Subsection 7.3.2, and compound splitting and joining in Subsection 7.3.3. Section 7.4 introduces the concept of multi-pass decoding, where additional processing steps are inserted after the decoding phase of the NMT decoder, such as dictionary-based unknown word replacement in Subsection 7.4.1 or hypothesis reranking in Subsection 7.4.2.1.

7.1 Implementation Details

Our experiments are based on the neural network framework `Blocks`[1,2] (Merriënboer et al. 2015), which itself is based on `Theano`[3] (Bergstra et al. 2010; Bastien et al. 2012). We trained our models on the high-performance cluster at Technische Universität Darmstadt, which imposes a hard-limit on the runtime of a job amounting to 24 hours. Hence, we had to pause the training process, save the model parameters to disk, and let a new job run that continues training from that point. For this, we implemented our blocks extension for checkpointing, i.e. an extension for saving and reloading the model. We also saved the model every 15 000 iterations in order to be able to monitor the progress of the model by calculating BLEU, METEOR, and TER scores over time. While it may be a good idea to try different hyperparameters, tuning for RNNs is not practical in our case given that it takes about one week to train a single hyperparameter configuration for MT on the WMT data set.

Therefore, we chose the same set of hyperparameters than Bahdanau, Cho, and Bengio (2014). Hyperparameters used in this work are given in Table 7.1. We did not use weight noise regularization or dropout. We also randomly shuffled the training data using a constant seed for all our experiments so that different models are trained on the same order of the data set.

We trained our models on a GPU and evaluated each checkpoint on a cluster node with 16 cores. We cannot give a precise number on the runtime for training because our training jobs were automatically distributed to hosts with either a Nvidia Tesla K20 or K40. We terminated the jobs at 60 000 iterations, which usually takes 20 hours, before it reaches the time limit. Since our data set contains 3 839 970 instances and we use a batch size of 80, a gingle epoch corresponds to around 48 000 iterations. Depending on the model, the best score on the development set was

[1] `https://github.com/mila-udem/blocks`
[2] `https://github.com/mila-udem/blocks-examples`
[3] `http://deeplearning.net/software/theano/`

usually achieved at around 400 000 to 500 000 iterations or 8 to 10 epochs, which took around 5 to 7 days. Nonetheless, training was continued until 800 000 iterations. Translating all sentences in the development set took around 24 hours on a machine with 16 cores.

Table 7.1: Hyperparameters used in our experiments.

Name	Value	Description
enc_nhids	1000	Number of hidden units in the encoder GRU
dec_nhids	1000	Number of hidden units in the decoder GRU
seq_len	50	Maximum sequence length
batch_size	80	Number of instances in a batch
step_rule	AdaDelta	Optimization step rule
src_vocab_size	30000	Source vocabulary size
trg_vocab_size	30000	Target vocabulary size

7.2 Development, Test Set, and the Baseline

In machine learning, it is essential to use separate data sets for development and testing purposes. If we were to use a single set for the selection of effective preprocessing or postprocessing techniques, overfitting onto this single data set becomes quite likely and could lead to subpar performance on unseen data. Hence, when we give scores on the test set, we acquire these scores by first maximizing the performance on the development set, and then we use the same configuration for predicting the test set. As we do minimal parameter tuning, in our case this just means we choose the same iteration for both the development and the test set.

As the baseline, we chose the vanilla model that additionally replaces unknown words in the target sentence by copying the aligned word in the source sentence. Copying already provides an increase of 2.8 BLEU points for English to German and 2.7 BLEU for German to English. This baseline was chosen instead of a baseline that does not copy because we believe it provides the fairest comparison when we evaluate our models that apply different strategies to counter the unknown word problem.

In Figure 7.1, we have also plotted BLEU, METEOR, and TER scores on the development and test set. It is not uncommon for two different set to show variance. Machine translation often works better for one data set than for another; hence, two distinct sets are not directly comparable. However, we can see in these plots that the trends are similar, meaning that the model provides the best score for either set at the same iteration. This confirms out choice of the development and test sets by showing that it works equally well on unseed data.

For the baseline model in Figure 7.1 and for English to German, the best BLEU score of 19.6 on the development set is produced at iteration 540 000, and the according test set score amounts to 19.2 BLEU. For German to English, translation performance on the development set is at its peak at iteration 555 000 at 23.3 BLEU with a corresponding test set score of 21.7 BLEU.

For the experiments in the following sections, we will apply the same concept of maximizing the BLEU score on the development set when providing scores for the test set. All scores in our experiments were computed on tokenized data.

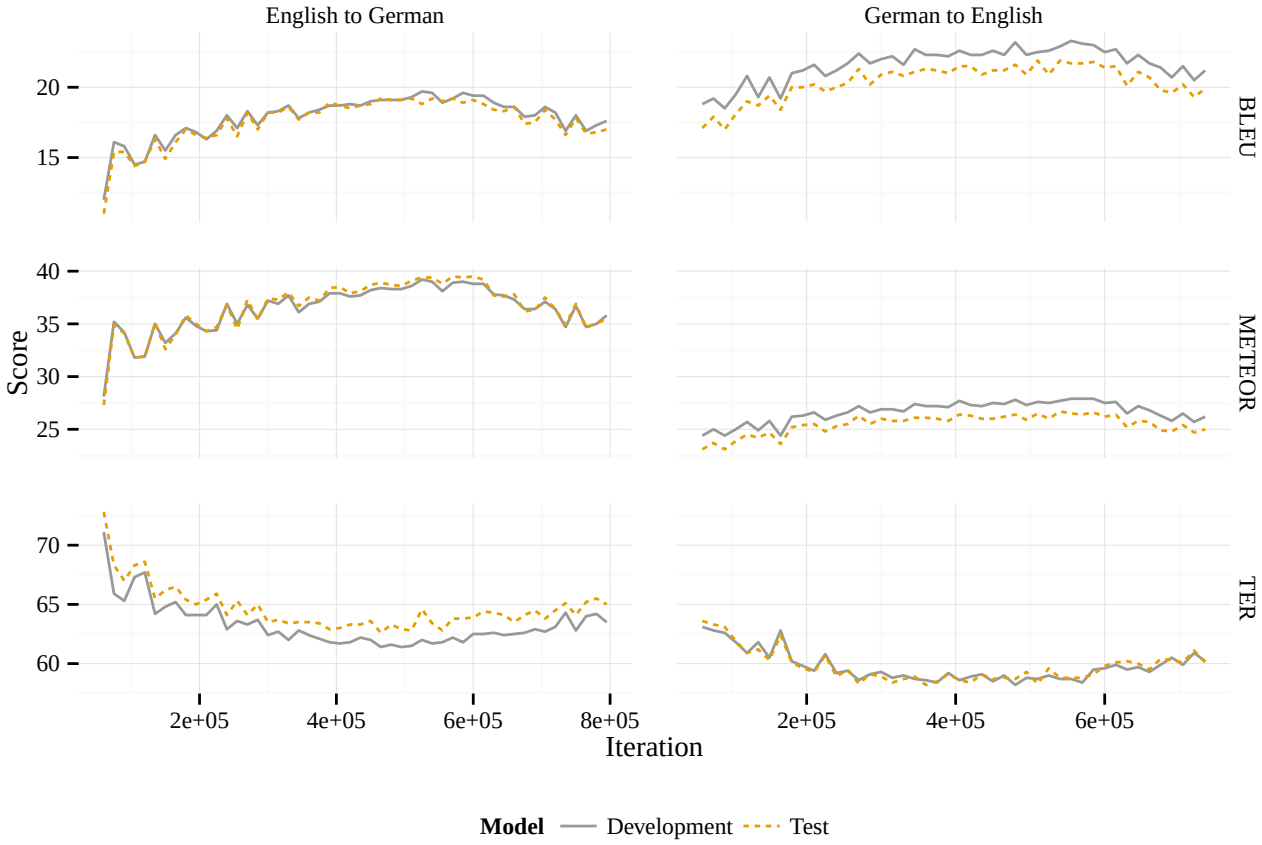

Figure 7.1: Comparison of evaluation scores on the development and test sets.

7.3 Preprocessing and Postprocessing Results

Preprocessing is a simple technique that tries to improve the translation performance by transforming the training and test data by a set of deterministic rules. It is often employed in SMT where it is known to increase the translation performance greatly. Up until now, NMT researchers have not focused on preprocessing and often perform no preprocessing at all. Due to this reason, we try to fill this gap in this section by providing a thorough analysis of different techniques.

7.3.1 Lowercasing

Lowercasing is one of the simplest preprocessing step possible, and we were unable to find similar experiments in NMT literature. By lowercasing the corpus, we effectively fit more words in our vocabulary because words are often found in a cased and uncased version. For instance, at the beginning of a sentence, a word starts with an uppercase letter while its starting letter is lowercase in another context. We lowercased both the source and target language and trained models for both translation directions.

As can be seen in Figure 7.2, for both translation directions we can see no increase in performance on the development set. We have also provided results on the test set in Table 7.2. While we can

Table 7.2: Results for lowercasing both the source and target language. Scores in subscripts are on the development set.

Translation Direction	Model	↑ BLEU^{uncased}	↑ METEOR^{uncased}	↓ TER^{uncased}
English → German	Baseline	$19.21_{19.60}$	$39.40_{39.00}$	$63.40_{61.70}$
	Lowercase	$19.25_{19.86}$	$39.40_{39.20}$	$63.30_{61.40}$
German → English	Baseline	$21.72_{23.30}$	$26.50_{27.90}$	$58.80_{58.70}$
	Lowercase	$22.33_{23.35}$	$27.10_{28.30}$	$58.50_{58.60}$

see marginal improvements for German → English on the test set, it should be noted that in this case, the best iteration on the development set just happens to match the best iteration on the test set and we do not think this is a representative gain given the other scores on the test set are well below this level.

It is important to note that because German nouns are capitalized, the lowercasing operation is performed more frequently for German than for English. We did not try to lowercase only the source language, but this might as well be a worthwhile experiment because the translation output is still cased; hence, we would not need to recase the system output using an additional layer in the translation system.

7.3.2 Replacement of Named Entities and Numbers

Because NMT models suffer from the OOV problem imposed by a fixed-sized vocabulary, we tried to replace NEs with a special *<UNK_PER>* token in the source as well as the target sentence. By doing so, correctly classified NEs are not added to the vocabulary, hence freeing up more space for words other than NEs.

This idea stems from our previous analysis of unknown words (see Table 5.2) in which we discovered that many unknown words are named entities. By replacing such words with a special token and not adding them to the vocabulary, we were expecting a performance increase in terms of machine translation evaluation metrics.

For POS tagging (see Section 3.1) and NER (see Section 3.3), we used the Stanford NLP suite[4] (Toutanova and Manning 2000; Toutanova et al. 2003).

An exemplary result of this technique is given by the following source sentence pair:

[4] `http://stanfordnlp.github.io/CoreNLP/`

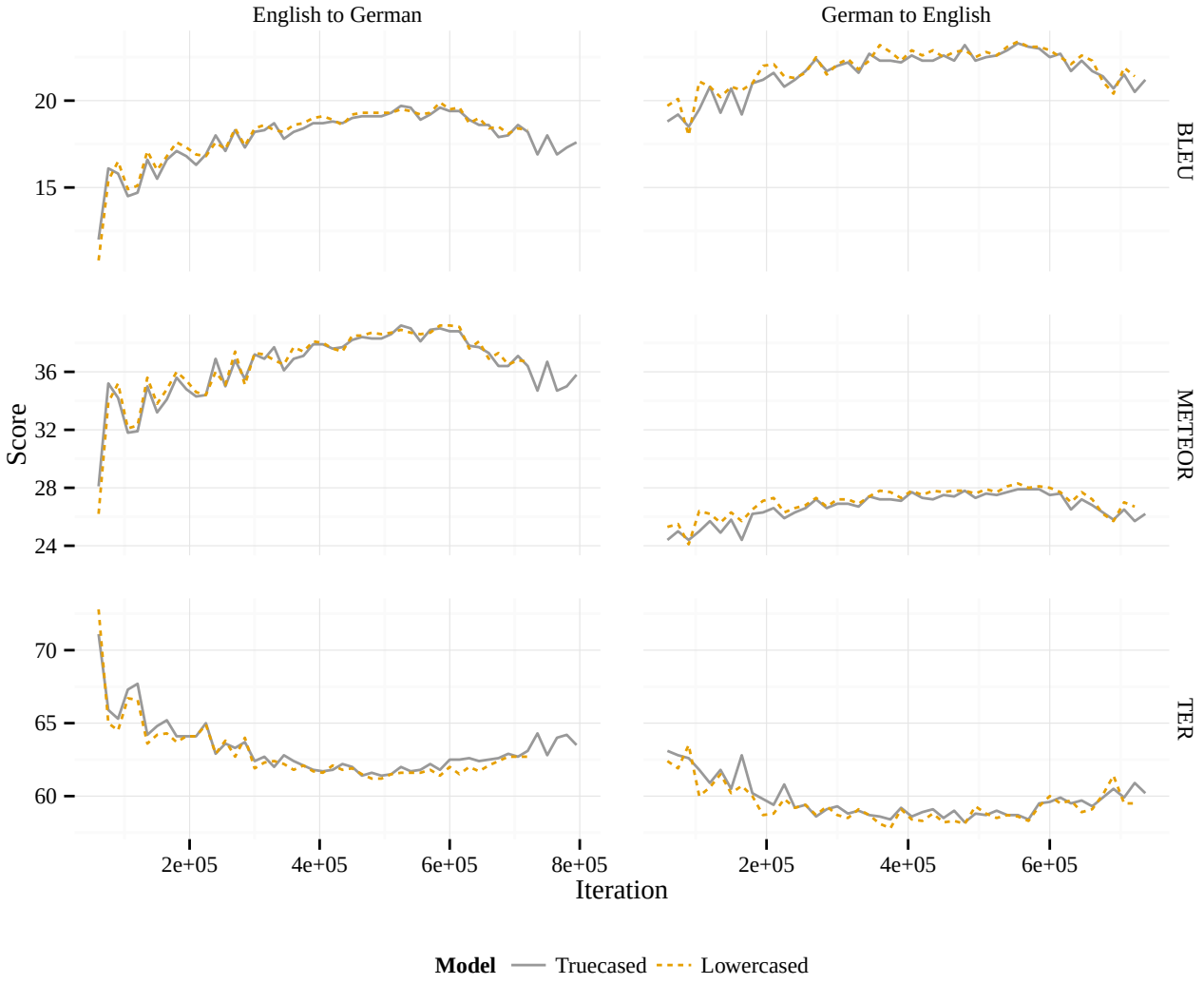

Figure 7.2: Effects of lowercasing. Lowercasing hardly contributes to a performance increase in NMT and may be avoided as doing so requires recasing the translation output.

Source sentence	I should like to comment on the speeches made Mr **Zimeray** and Mr **Laschet** and I will do so with all the diplomatic restraint for which I am famed, not least in Asia .
Processed source Sentence	I should like to comment on the speeches made Mr **<UNK_PER>** and Mr **<UNK_PER>** and I will do so with all the diplomatic restraint for which I am famed, not least in Asia.
Target Sentence	Ich möchte auf die Beiträge von Herrn **<UNK_PER>** und Herrn **<UNK_PER>** eingehen, und zwar mit einem Höchstmaß an diplomatischer Zurückhaltung, für die ich nicht zuletzt in Asien berühmt bin.
Processed Target sentence	Ich möchte auf die Beiträge von Herrn **Zimeray** und Herrn **Laschet** eingehen, und zwar mit einem Höchstmaß an diplomatischer Zurückhaltung, für die ich nicht zuletzt in Asien berühmt bin.

In this example, the names *Zimeray* and *Laschet* are replaced with *<UNK_PER>*. When the model translates this source sentence, it does generate the special tokens in the target sentence as well. Once the target sentence is generated, the special NE tokens are replaced according to the original input words, yielding the actual names.

We restricted replacements to persons only, as these are common NEs and show great variation. The rationale behind this idea is that the probability of a person's name in any context does not depend on who the person is. For example, the sentence *John likes Mary* is as likely as the sentence *Ben likes Mary*. This does not always hold true, however: the phrase *President Obama* should have a higher probability than *President McCain* because McCain was never president.

The same idea applies to another preprocessing step we tried: replacing numbers with a special *<UNK_NUM>* token. For this model, we replaced sequences of numbers with this token, also yielding words such as *<UNK_NUM>-year-old*. For evaluation, we replaced *<UNK_NUM>* tokens in the translation output using the alignment information produced by the model, and compared the scores to a baseline model where we replaced unknown words by copying from the source language.

The results of this experiment can be seen in Figure 7.3. For botch translation directions, the NE replacement model shows slight improvements early in the beginning of the training course, but then the model performs worse than the baseline at later iterations. Likewise, the number replacement model starts out much better than the baseline model, but then the performance drops greatly as training progresses. This results from severe overfitting to sentences that contain such special tokens. For instance, given a source sentence beginning with the words *In den frühen Abendstunden*, the model translates this sentence to *In the early <UNK_NUM>*, albeit no numbers are contained in the source sentence.

In our experimental settings, the replacement of named entities and numbers with special tokens seems to be ineffective in the long run It should be said that an alternative approach could be to use the more general *<UNK>* token instead of the specialized ones because then the model may be less likely to overfit. This approach could be generalized to a smarter vocabulary building technique where some words are left out on purpose. However, due to time constraints, we did not persuade this idea.

7.3.3 Compound Splitting and Joining

In this experiment, we tried to apply a corpus-based compound splitter as well as a hybrid corpus-based/linguistics-driven compound splitter to German. Compound splitting refers to the process of splitting up long German compound words into its compounds. For instance, *Jahresrückblick* is split into the words *Jahres* and *Rückblick*.

We have also described the use of filler letters in German compound words in Subsection 6.3.2. For example, the word *Jahresrückblick* is formed by the two compounds *Jahr* and *Rückblick*, yet between these two words the filler letters *es* are inserted. These filler letters are usually taken care of by compound splitters and completely removed. This is an acceptable strategy when translating from German to English because these letters fulfill no purpose other than connecting the compounds. Many compounds without filler letters can be translated directly into English, hence requiring no specific treatment.

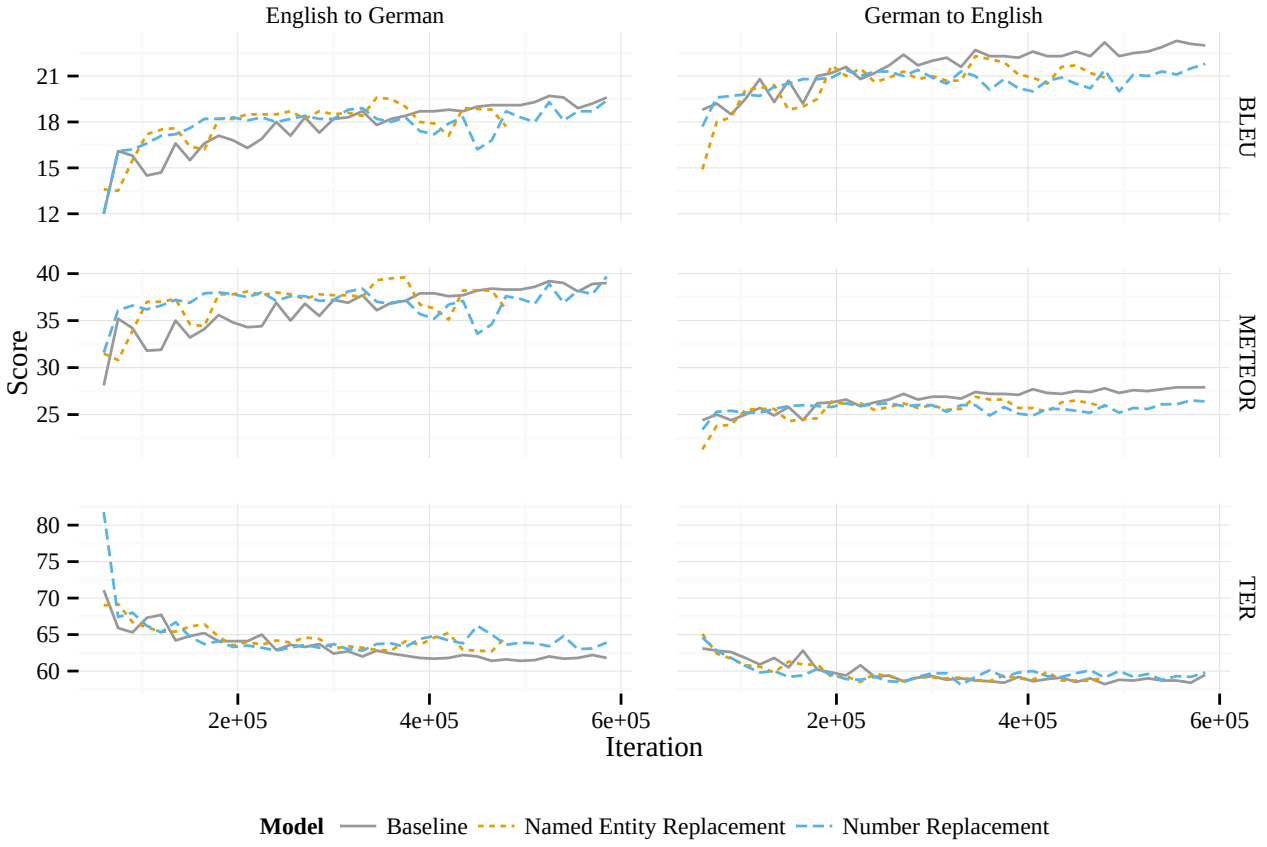

Figure 7.3: Effects of replacing named entities and numbers with special tokens.

However, the situation is not so clear when translating from English into German because filler letters are needed for reconstructing a correct compound word from its individual compounds. The next problem that arises when compound splitting is performed on the target side is that it requires us to join individual compounds together. We can deal with this problem by inserting special syntax into the training corpus that eases the process of compound merging. For example, instead of encoding the word *Autofahrer* (car driver) as *Auto* and *Fahrer*, we can encode it as *Auto @@ Fahrer*. By doing so, we hope the model will produce such connection markers at test time, which can then easily be joined during postprocessing. This also solves the problem of filler letters for German being on the target side, because filler letters can be encoded using this special syntax as well. For instance, the word *Kraftverkehrsverband* is made of up the words *Kraft*, *Verkehr*, and *Verband* and can be encoded as *Kraft @@ Verkehr @s@ Verband*.

This process has also been proposed by P. Williams et al. (2015) using the hybrid splitter by Fritzinger and Fraser (2010). We argue that this special syntax can also be produced by a much simpler corpus-driven splitter such as the IMS splitter. Hence, we modified the IMS splitter to output these filler letters between compound words. This modification applies to all compound splitters simply by comparing the splitter's output with its input and extracting filler letters using a regular expression.

Table 7.3: Results for source and target side compound splitting using either a corpus-based splitter or a hybrid corpus-based/linguistics-motivated one. In case of English to German translation, target side compound merging was performed. Scores in subscripts are on the development set.

Translation Direction	Model	↑ BLEU$^{\text{uncased}}$	↑ BLEU$^{\text{cased}}$	↑ METEOR$^{\text{uncased}}$	↓ TER$^{\text{uncased}}$
English → German	Baseline	$19.21_{19.60}$	$18.79_{19.11}$	$39.40_{39.00}$	$63.40_{61.70}$
	Hybrid Split	$19.34_{19.58}$	$18.95_{19.10}$	$39.90_{39.10}$	$63.70_{62.20}$
	Corpus-based Split	$19.41_{20.22}$	$18.99_{19.78}$	$40.30_{39.80}$	$64.40_{62.10}$
German → English	Baseline	$21.72_{23.30}$	$20.91_{22.36}$	$26.50_{27.90}$	$58.80_{58.70}$
	Hybrid Split	$22.30_{23.55}$	$21.37_{22.50}$	$27.30_{28.40}$	$60.00_{59.30}$
	Corpus-based Split	$24.42_{24.89}$	$23.41_{23.75}$	$29.80_{30.30}$	$58.20_{57.80}$

The results of our experiments on the development set can be seen in Figure 7.4, and the results on the test set are listed in Table 7.3. For English to German, both compound splitter show some performance increase, albeit the increase is rather small.

For German to English translation, the corpus-based splitting method works best, outperforming the baseline by 2.7 BLEU, 3.3 METEOR, and −0.6 TER. When inspecting the output of both splitters, we noticed that the hybrid/linguistic splitter produces a correct splitting more often. However, the corpus-based splitter provides better results for NMT. This contradiction has also been observed by Fritzinger and Fraser (2010) for phrase-based SMT. Corpus-driven methods tend to split more aggressively. For instance, consider the word *überall* which may get split into *über* and *all* by the corpus-driven approach due to these two compounds appearing very frequently. Contrariwise, the linguistic splitter is unlikely to split this word.

The explanation for a translation system performing better with aggressive splits could be that the system can recover from words that are split too much. This is especially applicable to NMT, where the problem of OOV words leads to many compound words not found in the vocabulary. Hence, the performance hit by unsplit words that may lead to OOV words is larger than the performance hit produced by overaggressive splitting.

We believe the reason compound splitting works much better for English than for German is that the vocabulary coverage for English is well below the coverage of German. As noted in Subsection 5.4.3, we found that only 68.73% German nouns are covered when encoding a text corpus using a fixed vocabulary of size 30 000, whereas English nouns are covered at a ratio of 93.25%. This led to our motivation behind an experiment that contains a compound splitter. Before we started training a NMT model with compound split data, we first explored the effects of compound splitting in terms of word coverage on the test data. For this analysis, we used the IMS Splitter (see Subsection 6.3.2) and our results can be seen in Figure 7.5, which shows that word coverage for German dramatically increases when compound splitting is applied.

When German is on the target side, the model successfully produces the special syntax we introduced that allows us to concatenate compound words as a postprocessing step. A translation of this model looks as follows:

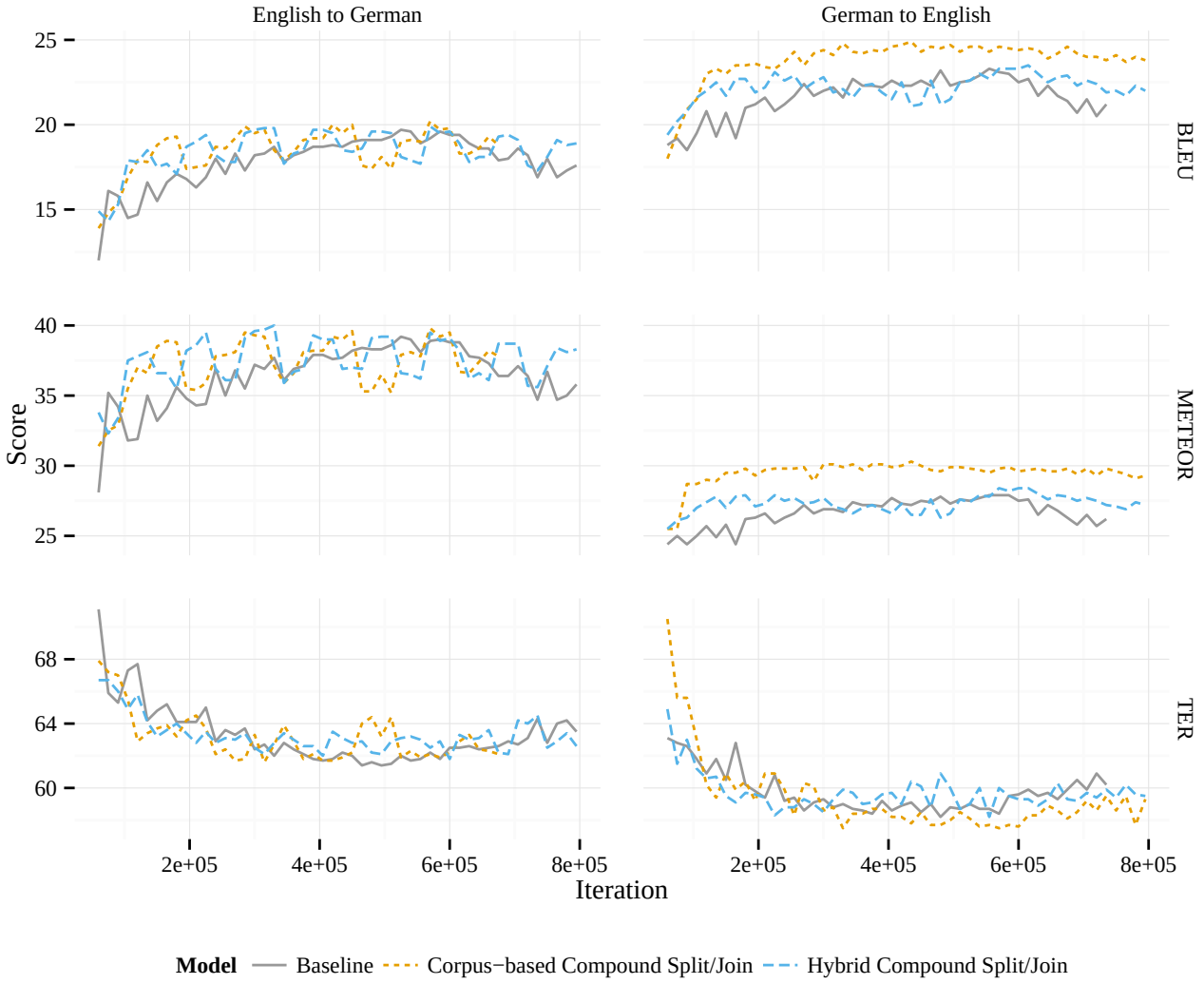

Figure 7.4: Effects of compound splitting and joining on the development set.

Source Sentence	The darker the meat, the higher the **pH value**.
Reference Sentence	Je dunkler das Fleisch, desto höher der **ph-Wert**.
Baseline Model	Je dunkler das Fleisch, desto höher der **pH**.
Split/Join Model	Je dunkler das Fleisch, desto höher der **pH @-@ Wert**.
Split/Join Model (merged)	Je dunkler das Fleisch, desto höher der **pH-Wert**.

The translation produced by the model for the phrase *pH value* is *pH @-@ Wert* and can just be concatenated by the use of a regular expression. This seems to work even with longer compound words, as shown by the following example:

In this case, the baseline model fails to translate the phrase altogether resulting in an unknown word. For readers not familiar with German, all sentences are correct translations for the source sentence when we disregard the unknown word, i.e. the baseline model and the split model differ in structure, but both are correct, and the split join model is more precisely in translating the word for road safety inspection.

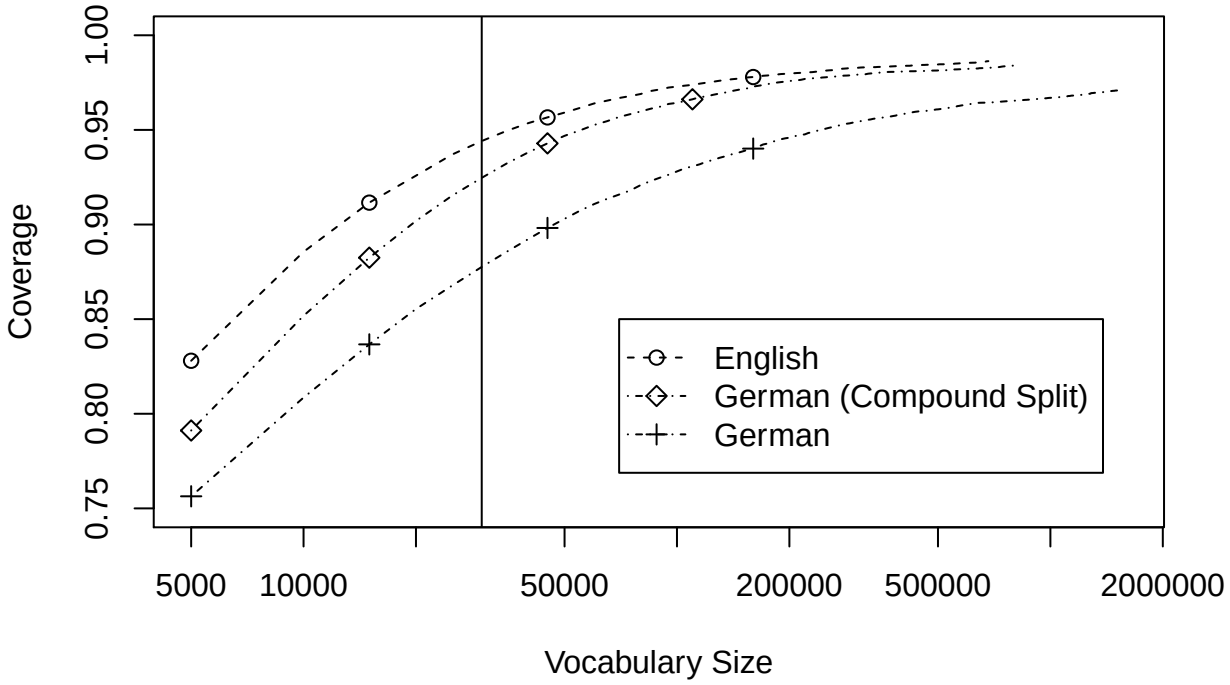

Figure 7.5: Coverage of the test set in relation to the vocabulary size on a logarithmic scale. The solid black line denotes the vocabulary size we use, i.e. 30 000 words yielding a coverage of 98.27% for English and 87.76% and 94.69% for German without and with compound splitting, respectively.

Another interesting observation is that the performance in terms of BLEU fluctuates wildly as splitting is applied. We compared the translation output between iterations showing favorable performance and iterations with a lower BLEU score. We found that sentences were often restructured causing the performance to drop. An example of this restructuring may look as follows:

Source Sentence	However, the new rules put in place will undoubtedly make it more difficult to exercise the right to vote in 2012 .
Reference Sentence	Allerdings werden die neu eingeführten Regeln im Jahr 2012 zweifellos die Ausübung des Wahlrechts erschweren .
Split/Join Model, Iteration 450 000	Mit den neuen Regeln wird es zweifellos schwieriger sein, das Wahlrecht im Jahr 2012 auszuüben .
Split/Join Model, Iteration 465 000	Die neuen Regeln werden jedoch zweifellos die Ausübung des Wahlrechts für 2012 erschweren .

The system output at iteration 450 000 in this example receives 35.29 BLEU points, whereas the output at iteration 465 000 receives 71.43 BLEU points. This discrepancy in BLEU is because BLEU scores sentences with more n-gram matches higher. The model's output at iteration 450 000 receives a 4-gram match for the phrase *zweifellos die Ausübung des Wahlrechts erschweren*, whereas

Source Sentence	A total of four **road safety inspections** were carried out.
Reference Sentence	Insgesamt seien vier **Verkehrsschauen** durchgeführt worden.
Baseline Model	Insgesamt wurden vier **<UNK>** durchgeführt.
Split/Join Model	Es wurden insgesamt vier **Straße @n@ Verkehr @s@ Inspektionen** durchgeführt.
Split/Join Model (merged)	Es wurden insgesamt vier **Straßenverkehrsinspektionen** durchgeführt.

the other model's output does not. Please do note that both outputs are perfect translations of the source sentence and would be judged as such by a human evaluator, stressing the importance of better machine learning evaluation metrics.

7.4 Multi-Pass Decoding

We describe computations performed after the NMT decoding phase has taken place as *multi-pass decoding*. This term is analogously used in ASR to rerank the hypothesis utterances according to features such as the LM score. These computations may include the replacement of unknown words, but also the reranking of hypothesis translations. Others may call the replacement of unknown words postprocessing, however since our dictionary-based replacement process produces new hypotheses itself that need some advanced selection technique such as reranking, we no longer call this concept postprocessing as there are more complex methods involved than simply using a one-to-one dictionary.

7.4.1 Addressing the Unknown Word Problem

As part of our multi-pass approach, we also experimented with dictionary-based replacement of unknown words. For this experiment, we prepared a human-created dictionary based on DictCC[5]. DictCC is a free, multilingual dictionary available for download in form of a tab separated text file and contains around one million entries. We parsed this dictionary using a regular expression to include source and target words as well as synonyms. Other fields in the dictionary not used include the POS and the context of the translation.

For each unknown word in the target sentence, in our multi-pass decoder, we looked up the single corresponding source word with the highest alignment weight, hence yielding a list of hypothesis words for each source word. Then, we create a list of possible translations incorporating all hypotheses for all unknown words. This is unlike Jean et al. (2014) and Luong et al. (2015) where the authors used an automatically generated dictionary using a word aligner, in which the final dictionary is only a 1-to-1 alignment between source and target word. The resulting hypotheses in our approach is equivalent to the Cartesian product, i.e. given a sentence with n unknown words $<\text{UNK}>_n$ and a set of hypotheses H_n for each unknown word, the finite n-ary product is given by:

$$H_1 \times \ldots \times H_n = \{(h_i, \ldots h_n) : h_i \in H_i\} \tag{7.1}$$

[5] `http://www.dict.cc/`

Figure 7.6: Dictionary-based replacement of unknown words.

In case an unknown target word is not found in the dictionary, we simply fall back to copying the aligned word from the source sentence. Due to the use of a manually created dictionary, we believe our fallback method works better because the dictionary contains less noise such as persons and organizations, which should most likely not be in the dictionary in the first place. And indeed, Jean et al. (2014) noticed that their approach works better when capitalized words are copied straight from the source sequence. We believe this improvement can be attributed to NEs starting with an uppercase letter, and not looking up NEs in a dictionary and instead copy them from the source sequence is more useful.

Having a list of hypothesis sentence at hand containing all words the cartesian product yields, we simply use a 4-gram LM for selecting the most likely sentence as final translation. For an introduction to LMs, please see Section 3.2. An example hypothesis list produced by our dictionary method looks as follows:

Source Sentence	Zwei Anlagen so nah beieinander: Absicht oder **Schildbürgerstreich**?
Reference Sentence	Two sets of lights so close to one another: intentional or just a **silly error**?
Baseline Model	Two systems so close together: intention or **<UNK>**?
Dict Hypothesis 1	Two systems so close together: intention or **cock-up**?
Dict Hypothesis 2	Two systems so close together: intention or **foolish act**?
Dict Hypothesis 3	Two systems so close together: intention or **piece of bungling**?

Table 7.4: Dictionary-based replacement of unknown words compared to a baseline model that copies unknown words from the source sentence, and a model that does not specifically deal with unknown words. Scores in subscripts are on the development set.

Translation Direction	Model	↑ BLEU^{uncased}	↑ BLEU^{cased}	↑ METEOR^{uncased}	↓ TER^{uncased}
English → German	Baseline	$19.21_{19.60}$	$18.79_{19.11}$	$39.40_{39.00}$	$63.40_{61.70}$
	No Replacement	$16.44_{17.70}$	$16.13_{17.32}$	$36.20_{36.80}$	$66.10_{63.50}$
	Dict Replacement	$19.17_{19.67}$	$18.80_{19.23}$	$39.70_{39.40}$	$63.30_{61.60}$
German → English	Baseline	$21.72_{23.30}$	$20.91_{22.36}$	$26.50_{27.90}$	$58.80_{58.70}$
	No Replacement	$18.97_{21.20}$	$18.32_{20.43}$	$24.20_{26.20}$	$61.30_{60.40}$
	Dict Replacement	$22.69_{24.21}$	$21.70_{23.22}$	$27.70_{29.00}$	$58.20_{58.10}$

It should be noted that a list of hypotheses is not the most compact way to represent candidates, and a word lattice might be better suited. A word lattice is a directed acyclic graph featuring a single start point. Each unknown word could be modeled as a node such that the graph becomes a directed graph with n nodes for each unknown word. Then, for each hypothesis, an edge is drawn to the next unknown word. We evaded performance problems by simply restricting our methods to sentences containing no more than five unknown words as we have observed that sequences containing more than that are usually enumeration of names, for which the fallback method of copying the source word works better. We also tried restricting our dictionary method to sentences containing no more than three words, but the performance considerably dropped as opposed to five words.

The results of the dictionary-based replacement process can be seen in Figure 7.6, in which we have also included a model that does not address the unknown word problem at all (No Replacement). For English to German, we see minimal improvements on the development set only. Contrariwise for German to English, we see considerable improvements for all three scores. We have also given test set scores in Table 7.4, which were acquired by maximizing the performance on the development and computing the scores on the test set at the same iteration. For translating into English, we see increases of 0.9 BLEU, 1.1 METEOR and −0.6 TER on the test set.

We believe that the reason our technique works much better for English to German than for German to English is that the vocabulary coverage of German is not as good as it is for English, as previously shown in Figure 7.5. Due to the poor vocabulary coverage of German, many English words that are indeed in the English vocabulary but not in the German vocabulary get translated to <*UNK*> tokens. As the uncovered German compound words are likely nouns formed by compounding, using a dictionary to look up the unknown words works very well.

Due to time constraints, we did not compare this method to by Luong et al. (2015). However, it should be noted that the authors treated NMT as a black box inserting special positional markers into the training data to yield alignments. This is not necessary for NMT with attention mechanism because word alignments are trained jointly. The researchers also limit their dictionary to 1-to-1 alignments, i.e. only the most frequently used translation is used for a dictionary entry, whereas we use a LM to select the most probable candidate words by maximizing the sentence probability through a LM.

7.4.2 Oracle Hypothesis and Reranking

As NMT makes use of beam search with a particular beam size (5 in our case), for each instance five hypotheses are generated. In basic NMT, the translation with the lowest cost given by the decoder is selected, right after receiving results from beam search. As we wanted to know how the other candidate sentences compare to the one selected by decoder cost mimimization, we computed the Oracle scores for our models.

An Oracle first computes a metric such as BLEU or METEOR on a sentence level for each hypothesis returned by beam search. Then, instead of minimizing the decoder cost, the best hypothesis is selected according to one of these metrics. As evaluation metrics are usually computed on a corpus level rather than e.g. an average of the sentence-level scores, the final BLEU, METEOR, or TER scores are calculated on a corpus level. The concept of Oracle scoring is well known in phrase-based SMT, but so far and as far as we are aware, NMT researchers have not yet considered Oracle scores when fine-tuning an end-to-end NMT system.

This brings us to another idea concerning the improvement of NMT. The hypotheses can be reranked by a machine learning model based on features such as sentence length ratio or the number of unknown words. There are multiple methods through which this can be achieved, and we concentrated on the following two:

- **Machine-Learned Ranking (MLR)**: The partial order induced by the sentence-level BLEU score is used to learn a model that predicts the rank of unseen hypotheses. The rank is then used to select the best hypothesis according to this model.
- **Regression**: A regression model for the sentence-level BLEU score is learned, possibly incorporating instance weights such that instances with many hypotheses produced by the dictionary method are not overrepresented.

We experimented with both approaches, and the results are presented in the next two sections. As far as the feature space is concerned, the following features were used for both methods:

- **cost**: The cost of the translation according to the NMT decoder.
- **lm_score**: The LM score of a 4-gram LM for the sentence.
- **source_length**: The number of tokens in the source sentence.
- **target_length**: The number of tokens in the target sentence.
- **length_ratio**: The number of source words divided by the number of target words.

7.4.2.1 Reranking by Machine-Learned Ranking

The data set we prepared for the first technique may look follows:

```
5 qid:1 1:2 2:1.1 3:-36.3408699036 4:4.6353597641 5:9 6:9.9
4 qid:1 1:2 2:1.1 3:-38.2936859131 4:5.1203827858 5:9 6:9.9
3 qid:1 1:1 2:1.2 3:-30.9083824158 4:5.19289159775 5:8 6:9.8
2 qid:1 1:1 2:1.1 3:-30.2387199402 4:5.2353348732 5:9 6:9.9
```

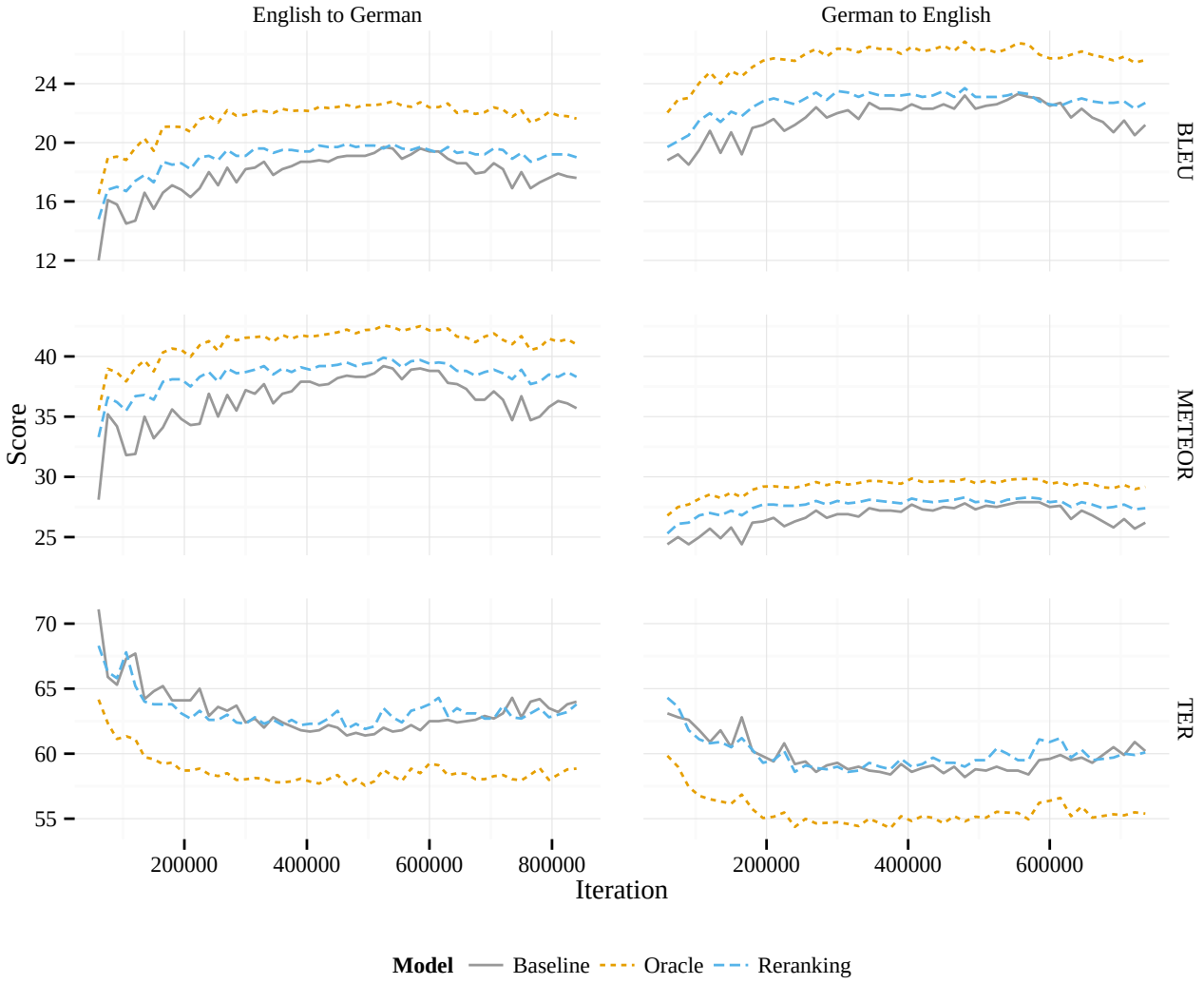

Figure 7.7: Performance of an oracle and a reranking model.

```
1 qid:1 1:2 2:1 3:-36.1737251282 4:5.82352828979 5:10 6:10
4 qid:2 1:1 2:1.21 3:-39.1920700073 4:8.19459724426 5:13 6:15.8
```

Where the first column corresponds to the rank, *qid* corresponds to the instance ID, i.e. *qid:1* refers to the first sentence in our data set to be translated. All following columns are features denoted by their indices.

For the second scheme, the data set looks as follows:

```
1 qid:1 1:2 2:1.1 3:-36.3408699036 4:4.6353597641 5:9 6:9.9
0 qid:1 1:0 2:1.1 3:-36.3408699036 4:4.6353597641 5:9 6:9.9
0 qid:1 1:2 2:1.1 3:-38.2936859131 4:5.1203827858 5:9 6:9.9
0 qid:1 1:1 2:1.2 3:-30.9083824158 4:5.19289159775 5:8 6:9.7
0 qid:1 1:1 2:1.1 3:-30.2387199402 4:5.2353348732 5:9 6:9.9
0 qid:1 1:0 2:1.2 3:-30.9083824158 4:5.19289159775 5:8 6:9.7
0 qid:2 1:1 2:1.21 3:-39.1920700073 4:8.19459724426 5:13 6:15.8
```

Where only the best hypothesis is marked as relevant by having a rank greater 0. For ranking, we used the ranking framework RankLib [6]. We trained our rankers as well as our regression model on the hypotheses produced by our NMT model at iteration 510 000 for the development set, and tested their performances on the test set for the same iteration. We compared Multiple Additive Regression Trees (MART) by Friedman (2001), AdaRank by Xu and Li (2007), RankNet by Burges et al. (2005), and Random Forest (RF) by Breiman (2001). All rankers were optimized using the Expected Reciprocal Rank (ERR) metric by Chapelle et al. (2009) through gridsearch.

We tried the above rankers for both schemes, but neither scheme or ranker increased the performance of the translation output. Hence, we conclude that ranking for using the named rankers is ineffective.

7.4.2.2 Reranking by Regression

When shifting our experiments from ranking to regression, we saw marginal benefit concerning translation performance. We first trained regression models for BLEU on the development set using 10-fold cross-validation and grid search, and then used this model to compute the BLEU scores on the test set. Afterward, we used the predicted BLEU score for each hypothesis to finally select the best candidate for each instance. The final BLEU score was then calculated based on all sentences and compared to a baseline model where no reranking has taken place.

We repeated this experiment for each model we had saved, which yields the results in Figure 7.7. The Oracle scores in this graph are only for reference and the algorithms we tested for reranking include Multivariate Adaptive Regression Splines (MARS) by Friedman (1991) and Least Angle Regression (LARS) by Efron et al. (2004). In Table 7.5 we selected the best performing iteration on the development set, and then used this very iteration for predicting the test set.

For English to German, the reranking model performs consistently better than the baseline on the development set, yet the performance worsened when we computed scores on the test set. For German on English, the improvement on the test set amounts to 0.3 BLEU, and 0.2 METEOR for MARS. It is interesting that the TER score worsened using any technique, and we believe this is due to the reranker preferring longer sentence, which in case of unmatched tokens results in a higher TER. One of the most important features used by both MARS and LARS are the length of the source and target sentence. The LM score plays a minor role, and the results are only slightly affected when the LM score is not used.

For German to English, we tried the same experiments on top of our model that makes use of compound splitting, interestingly the scores worsened. We believe this is because reranking works best on language pairs that have unequal sentence lengths and possibly one-to-many word alignments, as it is the case with unprocessed German compared to English. Once we preprocess German and split compound words, the length of sentences in German is much closer to the length of English sentences. This has important consequences for NMT, as it means that when source and target languages are different in average sentence length, this difference should be taken into account when selecting the best hypothesis. It may not be necessary to learn a reranking model the way we did. Instead, it might be enough to normalize the score of the decoder by the length of the sentence, and we think further research is required in this direction.

[6] `https://people.cs.umass.edu/~vdang/ranklib.html`

Table 7.5: Results of reranking translation hypotheses by regression. Regression models were learned on the development data using 10-fold cross-validation and then used to predict the BLEU score on the hypotheses in the test set. The hypotheses for which the model predicted the highest BLEU score was then selected to compute the given scores. Scores in subscript are on the development set.

Translation Direction	Model	↑ BLEU^{uncased}	↑ METEOR^{uncased}	↓ TER^{uncased}
English → German	Baseline	$19.2_{19.6}$	$39.4_{39.0}$	$63.4_{61.7}$
	MARS	$18.7_{19.9}$	$40.1_{39.7}$	$65.4_{62.8}$
	LARS	$19.0_{19.9}$	$40.1_{39.6}$	$64.8_{62.9}$
German → English	Baseline	$21.7_{23.3}$	$26.5_{27.9}$	$58.8_{58.7}$
	MARS	$21.1_{23.5}$	$26.8_{28.2}$	$59.7_{59.5}$
	LARS	$22.0_{23.4}$	$26.7_{28.1}$	$59.5_{59.5}$

7.5 Pre-Epoch Training Data Reshuffling

An often successful method to improve the performance of NNs, in general, is to reshuffle the training data after every epoch. Hence, we stopped the training process briefly once we have gone through an epoch (an entire run through all instances in the training data), shuffled the training data randomly, and resumed training thereafter.

The results of this process can be inspected in Figure 7.8. The first thing that comes to mind when analyzing these results is that the reshuffled model becomes more unstable. However, contrary to our belief, this is not the case and is rather due to the way our evaluation method works. We saved and evaluated all model every 15000 iterations, but for the reshuffling model it was necessary to save it additionally after every epoch. As it turns out, the many spikes that can be seen in the reshuffle model graph are at the end of each epoch. Hence, we can conclude that the best performance is usually given *after* training an entire epoch. We believe this also applies to the other models.

For this reason, it is hard for us to compare the shuffled model against the baseline because for the baseline we have not saved the model at the end of an epoch.

However, for iteration $\in [400000, 60000]$ we can see an average improvement of 0.69 BLEU, 0.77 METEOR, and −0.52 TER on the dev set. On the train set the improvements are 0.70 BLEU, 0.84 TER, and −0.68 TER. Reshuffling after each epoch is highly effective and contributes greatly to the overall performance of the translation system.

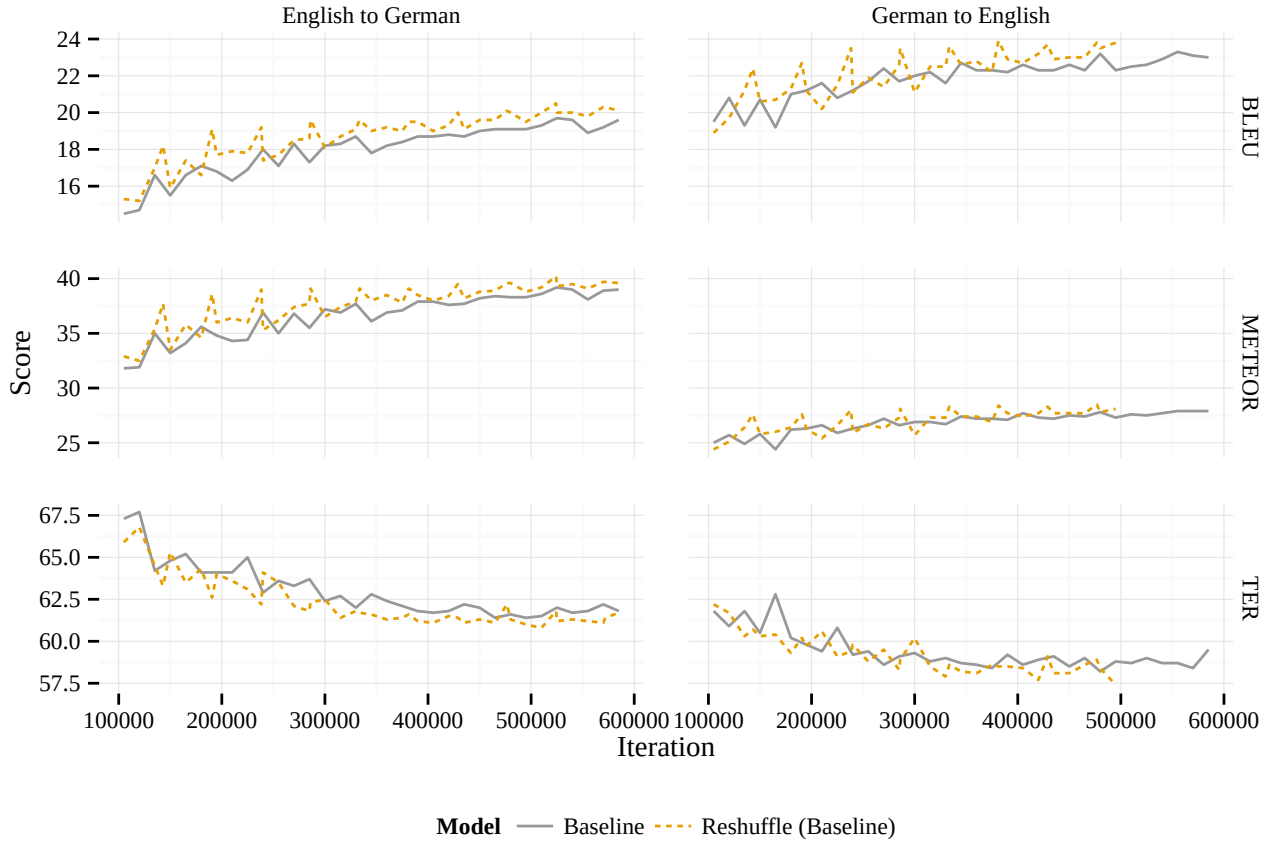

Figure 7.8: Effects of reshuffling the training data after every epoch. The spikes that are particularly present in the reshuffled model are at the end of an epoch.

7.6 Results Summary

In the previous sections, we tried to increase the performance of NMT by lowercasing, replacing NEs and numbers with special markers, compound splitting and joining, dictionary-based unknown word replacement, reranking the translation hypotheses, and pre-epoch data set reshuffling.

The scores for our most promising two techniques can be found in Table 7.6. For English to German translation, the gain achieved through target side compound splitting and joining amounts to 0.2 BLEU, 0.9 METEOR. It is noteworthy to state that our model successfully forms German compound words on the target side, whereas the baseline model often fails to translate multi-word English phrases.

For German to English, we saw much greater gains. For dictionary-based word replacement, the scores increased by 0.97 BLEU, 1.2 METEOR, and −0.6 TER. For compound splitting, the increases amount to 2.7 BLEU, 3.3 METEOR, and −0.6 TER. When we combined these two techniques in Table 7.6, the scores improved even further.

As far as reranking translation hypotheses is concerned, it can be said that while our ideas looked promising on the development set, they were not reproducible on the test set.

Table 7.6: Results for source and target side compound splitting and dictionary-based unknown word replacement. The combined model applies these two techniques at the same time. Scores in subscripts are on the development set.

Translation Direction	Model	↑ BLEU^{uncased}	↑ BLEU^{cased}	↑ METEOR^{uncased}	↓ TER^{uncased}
English → German	Baseline	$19.21_{19.60}$	$18.79_{19.11}$	$39.40_{39.00}$	$63.40_{61.70}$
	Corpus-based Split	$19.41_{20.22}$	$18.99_{19.78}$	$40.30_{39.80}$	$64.40_{62.10}$
	Dict Replacement	$19.17_{19.67}$	$18.80_{19.23}$	$39.70_{39.40}$	$63.30_{61.60}$
	Split + Dict	$19.33_{20.18}$	$18.94_{19.75}$	$40.30_{40.00}$	$64.50_{62.00}$
German → English	Baseline	$21.72_{23.30}$	$20.91_{22.36}$	$26.50_{27.90}$	$58.80_{58.70}$
	Corpus-based Split	$24.42_{24.89}$	$23.41_{23.75}$	$29.80_{30.30}$	$58.20_{57.80}$
	Dict Replacement	$22.69_{24.21}$	$21.70_{23.22}$	$27.70_{29.00}$	$58.20_{58.10}$
	Split + Dict	$24.60_{25.20}$	$23.54_{24.02}$	$30.10_{30.70}$	$58.10_{57.60}$

Our results also show that reshuffling of the training data before each epoch is highly beneficial to NMT, but due to time constraints we were unable to pursue this method further.

8 Conclusion and Future Work

Neural network based learning algorithms are sometimes thought to require minimal preprocessing, yet in this study, we have shown that preprocessing cannot be neglected altogether when using current word-based NMTs architectures. While we had no success with simple preprocessing such as lowercasing, we obtained substantial improvements of 2.7 BLEU points for German to English translations by splitting German compound words on the source side. We also applied compound splitting when German was on the target side by modifying the compound splitter's output to contain special syntax that can later be used to create compound words from their individual components. While we saw only a modest improvement of 0.2 BLEU, it allows the model to merge German words on the target side, which the baseline model is unable to perform.

As word-based NMT is subject to the OOV problem that leads to unknown words not being translated, we also explored several mechanisms to replace unknown words on the target side. The attention mechanism allows us to jointly learn to align and translate, and the alignment weights can be used to find corresponding source words for a given target word. Based on the alignment weights, we tried to copy the aligned source word for each unknown target word, yielding a dramatic increase of 2.75 BLEU over a model that does not replace unknown target words at all. We also used a manually created dictionary[1] to select candidate words and jointly maximized the probability of a sentence appearing with these words in place using a language model, resulting in a considerable increase of 0.97 BLEU for German to English. We believe a dictionary approach is beneficial when the source language is poorly covered by its vocabulary since we could not observe such improvements when translating English sentences to German ones. In our analysis of the OOV problem, we found that named entities are poorly covered by a fixed-size dictionary, but in many cases this does not pose a problem because these can simply be copied from the source sentence.

For future work, we believe that the treatment of words as atomic units in NMT is seriously hindering the progression of its performance. This was partially shown in our study because the performance has increased when compound splitting was applied. The reason for this problem is two-fold. First, the use of a fixed-size vocabulary may not be an issue for languages such as English in which multi-word phrases are written as space-separated tokens and thus easily encoded. But for languages such as German, Finish, and Russian these words are often not found in the dictionary because word formation leads to an explosion in vocabulary size. Second, we believe 1-to-many or many-to-1 word alignments are often not translated correctly even if the compound word is present in the vocabulary. Compound words are, however, not the only case where an explosion in vocabulary size can happen. During our analysis of the OOV problem, we showed that for German, the coverage of finite verbs, participle perfect, and infinitive also leave much to be desired. One possible solution could be to treat characters as atomic units, and this has been tried in e.g. Ling et al. (2015), but the performance is not yet on par with word-based

[1] `http://dict.cc`

NMT. Instead of encoding sentences through words or characters, it is possible to encode them as subword units. Based on the idea that various word classes are translatable via smaller units than words, Sennrich, Haddow, and Birch (2015) propose to use such units. Indeed, the individual compounds in German compound words can be seen as such subword units, but this idea can be generalized further. This is also specifically applicable to languages like Russian, where words are written differently depending on the gender of the object in a sentence.

As the training of NMT models takes a lot of time, we hope also to see advances in a way to reduce the training time of such models. Training is already exclusively performed on GPUs instead of CPUs, but the use of more than one GPU or the distribution to multiple machines poses a problem. Large corporations such as Google (Abadi et al. 2015) and Microsoft (Agarwal et al. 2014) also seem to have a desire for faster training time of such models. Zhang, Choromanska, and LeCun (2015) introduce Elastic Averaging SGD (EASGD) for training deep neural networks in the stochastic setting that allows for parallelized training over multiple GPUs.

9 Glossary

ASR Automatic Speech Recognition. 10, 54, 76, 86

BLEU Bilingual Evaluation Understudy. 24, 25, 51, 66, 67, 73, 75, 78, 79, 81–83, 85, 86

BPTT Backpropagation Through Time. 38, 39, 51, 86

BRNN Bidirectional Recurrent Neural Network. 36, 38, 51, 54, 86

CAT Computer-Assisted Translation. 8, 86

CRF Conditional Random Fields. 21, 86

EOS End-of-Sequence. 49, 50, 86

ERR Expected Reciprocal Rank. 81, 86

FNN Feedforward Neural Network. 33, 35–37, 51, 53, 86

GRU Gated Recurrent Unit. 30, 41, 43–46, 51, 86

HMM Hidden Markov Model. 15, 17, 18, 20, 21, 41, 45, 46, 86

LARS Least Angle Regression. 81, 86

LM Language Model. 20, 21, 76–79, 81, 86

LSTM Long Short-Term Memory. 13, 30, 41, 43–46, 48, 51, 86

MAP Mean Average Precision. 86

MARS Multivariate Adaptive Regression Splines. 81, 86

MART Multiple Additive Regression Trees. 81, 86

METEOR Metric for Evaluation of Translation with Explicit ORdering. 26, 27, 66, 67, 73, 78, 79, 81–83, 86

MILA Montreal Institute for Learning Algorithms. 86

MLP Multilayer Perceptron. 86

MLR Machine-Learned Ranking. 79, 86

MT Machine Translation. 8, 9, 15, 24, 26, 27, 29, 54, 63, 66, 86

NE Named Entity. 14, 21, 56, 57, 66, 69, 71, 77, 83, 86

NER Named Entity Recognizer. 14, 21, 69, 86

NLP Natural Language Processing. 8, 9, 14, 69, 86

NMT Neural Machine Translation. 9, 10, 12, 13, 21, 22, 30, 37, 41, 43, 44, 46–48, 50, 51, 53–55, 58, 60, 63, 66, 68–70, 73, 76, 78, 79, 81, 83–86, 97

NN Neural Network. 30, 35, 37, 39, 46, 82, 86

OOV Out-of-Vocabulary. 6, 9, 10, 12, 54–56, 69, 73, 85, 86

POS Part of Speech. 14, 15, 17, 19, 20, 55–57, 64, 69, 76, 86

RF Random Forest. 81, 86

RNN Recurrent Neural Network. 9, 15, 30, 33, 35–39, 41–46, 48–54, 58, 66, 86

SGD Stochastic Gradient Descent. 38, 86

SMT Statistical Machine Translation. 8–10, 13, 22, 23, 30, 47, 51, 58, 64, 68, 73, 79, 86

TER Translation Error Rate. 25, 66, 67, 73, 78, 79, 81–83, 86

WSD Word-Sense Disambiguation. 9, 86

Bibliography

Abadi, Martin, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems* (cited on p. 86).

Agarwal, Amit, Eldar Akchurin, Chris Basoglu, Guoguo Chen, Scott Cyphers, Jasha Droppo, Adam Eversole, Brian Guenter, Mark Hillebrand, Ryan Hoens, Xuedong Huang, Zhiheng Huang, Vladimir Ivanov, Alexey Kamenev, Philipp Kranen, Oleksii Kuchaiev, Wolfgang Manousek, Avner May, Bhaskar Mitra, Olivier Nano, Gaizka Navarro, Alexey Orlov, Marko Padmilac, Hari Parthasarathi, Baolin Peng, Alexey Reznichenko, Frank Seide, Michael Seltzer, Malcolm Slaney, Andreas Stolcke, Yongqiang Wang, Huaming Wang, Kaisheng Yao, Dong Yu, Yu Zhang, and Geoffrey Zweig (2014). *An Introduction to Computational Networks and the Computational Network Toolkit*. Tech. rep. MSR-TR-2014-112 (cited on p. 86).

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473* (cited on pp. 12, 30, 48, 51, 54, 66).

Banerjee, Satanjeev and Alon Lavie (2005). METEOR: An Automatic Metric for Machine Translation Evaluation with Improved Correlation with Human Judgments. *Proceedings of the Association for Computational Linguistics (ACL) Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. Vol. 29, pp. 65–72 (cited on p. 26).

Bastien, Frédéric, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio (2012). Theano: New Features and Speed Improvements. *arXiv preprint arXiv:1211.5590* (cited on p. 66).

Baum, Leonard and Ted Petrie (1966). Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, pp. 1554–1563 (cited on p. 15).

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Janvin (2003). A Neural Probabilistic Language Model. *The Journal of Machine Learning Research* 3, pp. 1137–1155 (cited on p. 21).

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi (1994). Learning Long-Term Dependencies with Gradient Descent is Difficult. *Transactions on Neural Networks* 5.2, pp. 157–166 (cited on p. 39).

Bergstra, James, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio (2010). Theano: A CPU and GPU Math Expression Compiler. *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Vol. 4, p. 3 (cited on p. 66).

Bishop, Christopher (2006). Pattern Recognition and Machine Learning. Springer (cited on p. 30).

Breiman, Leo (2001). Random Forests. *Machine Learning* 45.1, pp. 5–32 (cited on p. 81).
Bridle, John (1990). Training Stochastic Model Recognition Algorithms as Networks Can Lead to Maximum Mutual Information Estimation of Parameters. *Advances in Neural Information Processing Systems (NIPS)*, pp. 211–217 (cited on p. 53).
Britz, Denny (2015). *Recurrent Neural Network Tutorial*. URL: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns (visited on 12/17/2015) (cited on p. 30).
Burges, Chris, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender (2005). Learning to Rank Using Gradient Descent. *Proceedings of the 22nd International Conference on Machine Learning (ICML)*. Bonn, Germany, pp. 89–96 (cited on p. 81).
Chapelle, Olivier, Donald Metlzer, Ya Zhang, and Pierre Grinspan (2009). Expected Reciprocal Rank for Graded Relevance. *Proceedings of the 18th Association for Computing Machinery Conference on Information and Knowledge Management (CIKM)*. Hong Kong, China, pp. 621–630 (cited on p. 81).
Cho, Kyunghyun (2015). *Introduction to Neural Machine Translation with GPUs*. URL: http://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-with-gpus (visited on 12/17/2015) (cited on p. 30).
Cho, Kyunghyun, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio (2014a). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *arXiv preprint arXiv:1409.1259* (cited on pp. 12, 51).
Cho, Kyunghyun, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014b). Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv preprint arXiv:1406.1078* (cited on pp. 12, 30, 41, 43, 48, 54).
Chung, Junyoung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv preprint arXiv:1412.3555* (cited on pp. 43, 44).
– (2015). Gated Feedback Recurrent Neural Networks. *arXiv preprint arXiv:1502.02367* (cited on p. 44).
Clark, Jonathan, Chris Dyer, Alon Lavie, and Noah Smith (2011). Better Hypothesis Testing for Statistical Machine Translation: Controlling for Optimizer Instability. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*. Vol. 2. Portland, Oregon, USA, pp. 176–181 (cited on p. 25).
Denkowski, Michael and Alon Lavie (2010). Meteor-Next and the Meteor Paraphrase Tables: Improved Evaluation Support for Five Target Languages. *Proceedings of the 5th Joint Workshop on Statistical Machine Translation and MetricsMATR (WMT-MetricsMATR)*. Uppsala, Sweden, pp. 339–342 (cited on p. 26).
– (2014). Meteor Universal: Language Specific Translation Evaluation for Any Target Language. *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL) Workshop on Statistical Machine Translation (WMT)*. Gothenburg, Sweden (cited on p. 26).
Devlin, Jacob, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul (2014). Fast and Robust Neural Network Joint Models for Statistical Machine Translation. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*. Baltimore, Maryland, pp. 1370–1380 (cited on p. 12).

Duh, Kevin and Katrin Kirchhoff (2008). Beyond Log-Linear Models: Boosted Minimum Error Rate Training for N-Best Re-Ranking. *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL) on Human Language Technologies: Short Papers*. Columbus, Ohio, USA, pp. 37–40 (cited on p. 13).

Efron, Bradley, Trevor Hastie, Iain Johnstone, and Robert Tibshirani (2004). Least Angle Regression. *The Annals of Statistics* 32.2, pp. 407–499 (cited on p. 81).

Faruqui, Manaal and Sebastian Padó (2010). Training and Evaluating a German Named Entity Recognizer with Semantic Generalization. *Proceedings of the "Konferenz zur Verarbeitung natürlicher Sprache" (German)*. Saarbrücken, Germany, pp. 129–133 (cited on p. 21).

Finkel, Jenny Rose, Trond Grenager, and Christopher Manning (2005). Incorporating Non-Local Information Into Information Extraction Systems by Gibbs Sampling. *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*. Ann Arbor, Michigan, USA, pp. 363–370 (cited on p. 21).

Friedman, Jerome (1991). Multivariate Adaptive Regression Splines. *The Annals of Statistics*, pp. 1–67 (cited on p. 81).

– (2001). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, pp. 1189–1232 (cited on p. 81).

Fritzinger, Fabienne and Alexander Fraser (2010). How to Avoid Burning Ducks: Combining Linguistic Analysis and Corpus Statistics for German Compound Processing. *Proceedings of the 5th Joint Workshop on Statistical Machine Translation and MetricsMATR (WMT-MetricsMATR)*. Uppsala, Sweden, pp. 224–234 (cited on pp. 13, 63, 64, 72, 73).

Gers, Felix and Jürgen Schmidhuber (2000). Recurrent Nets That Time and Count. *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. Vol. 3. Como, Italy: IEEE, pp. 189–194 (cited on p. 43).

Goodfellow, Ian, Aaron Courville, and Yoshua Bengio (2015). Deep Learning. Book in preparation for MIT Press (cited on p. 30).

Graves, Alan, Abdel-rahman Mohamed, and Geoffrey Hinton (2013). Speech Recognition with Deep Recurrent Neural Networks. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. Vancouver, Canada: IEEE, pp. 6645–6649 (cited on p. 45).

Graves, Alex (2013). Generating Sequences with Recurrent Neural Networks. *arXiv preprint arXiv:1308.0850* (cited on p. 45).

Graves, Alex, Marcus Liwicki, Santiago Fernandez, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber (2009). A Novel Connectionist System for Unconstrained Handwriting Recognition. *Transactions on Pattern Analysis and Machine Intelligence* 31.5, pp. 855–868 (cited on p. 44).

Graves, Alex, Greg Wayne, and Ivo Danihelka (2014). Neural Turing Machines. *arXiv preprint arXiv:1410.5401* (cited on pp. 46, 54).

Greff, Klaus, Rupesh Kumar Srivastava, Jan Koutnik, Bas R Steunebrink, and Jürgen Schmidhuber (2015). LSTM: A Search Space Odyssey. *arXiv preprint arXiv:1503.04069* (cited on p. 43).

Heafield, Kenneth, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn (2013). Scalable Modified Kneser-Ney Language Model Estimation. *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL)*. Sofia, Bulgaria, pp. 690–696 (cited on p. 21).

Hochreiter, Sepp (1991). Untersuchungen Zu Dynamischen Neuronalen Netzen. German. PhD thesis. Technische Universität München (cited on p. 39).

Hochreiter, Sepp and Jürgen Schmidhuber (1997). Long Short-Term Memory. *Neural Computation* 9.8, pp. 1735–1780 (cited on pp. 12, 41).

Jean, Sébastien, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio (2014). On Using Very Large Target Vocabulary for Neural Machine Translation. *arXiv preprint arXiv:1412.2007* (cited on pp. 12, 76, 77).

Jurafsky, Dan and James Martin (2000). Speech & Language Processing. Pearson (cited on p. 21).

Kneser, Reinhard and Hermann Ney (1995). Improved Backing-Off for M-Gram Language Modeling. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. Vol. 1. IEEE. Detroit, Michigan, USA, pp. 181–184 (cited on p. 21).

Knight, Kevin (1999). Decoding Complexity in Word-Replacement Translation Models. *Computational Linguistics* 25.4, pp. 607–615 (cited on p. 23).

Koehn, Philipp (2009). Statistical Machine Translation. Cambridge University Press (cited on pp. 21–23).

Koehn, Philipp and Kevin Knight (2003). Empirical Methods for Compound Splitting. *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*. Budapest, Hungary, pp. 187–193 (cited on p. 64).

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. (2007). Moses: Open Source Toolkit for Statistical Machine Translation. *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*. Prague, Czech Republic, pp. 177–180 (cited on pp. 25, 59).

Lafferty, John, Andrew McCallum, and Fernando Pereira (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. *Proceedings of the 18th International Conference on Machine Learning (ICML)*, pp. 282–289 (cited on p. 21).

Le, Quoc, Navdeep Jaitly, and Geoffrey Hinton (2015). A Simple Way to Initialize Recurrent Networks of Rectified Linear Units. *arXiv preprint arXiv:1504.00941* (cited on p. 41).

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). Deep Learning. *Nature* 521, pp. 436–444 (cited on p. 30).

LeCun, Yann, Léon Bottou, Genevieve Orr, and Klaus-Robert Müller (2012). Efficient Backprop. *Neural networks: Tricks of the Trade*. Springer, pp. 9–48 (cited on p. 32).

Ling, Wang, Isabel Trancoso, Chris Dyer, and Alan Black (2015). Character-Based Neural Machine Translation. *arXiv preprint arXiv:1511.04586* (cited on p. 85).

Luong, Minh-Thang, Hieu Pham, and Christopher Manning (2015). Effective Approaches to Attention-Based Neural Machine Translation. *arXiv preprint arXiv:1508.04025* (cited on p. 54).

Luong, Minh-Thang, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba (2015). Addressing the Rare Word Problem in Neural Machine Translation. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL) and the 7th International Joint Conference on Natural Language Processing (IJCNLP)*. Beijing, China (cited on pp. 12, 55, 76, 78).

Machácek, Matouš and Ondrej Bojar (2014). Results of the Wmt14 Metrics Shared Task. *Proceedings of the 9th Workshop on Statistical Machine Translation (WMT)*, pp. 293–301 (cited on pp. 24, 27, 28).

Manning, Christopher (2011). Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics? *Computational Linguistics and Intelligent Text Processing*. Springer, pp. 171–189 (cited on p. 20).

Marcus, Mitchell, Mary Marcinkiewicz, and Beatrice Santorini (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics* 19.2, pp. 313–330 (cited on p. 15).

McCulloch, Warren and Walter Pitts (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics* 5.4, pp. 115–133 (cited on p. 30).

Merriënboer, Bart van, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio (2015). Blocks and Fuel: Frameworks for Deep Learning. *arXiv preprint arXiv:1506.00619* (cited on p. 66).

Miller, George, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller (1990). Introduction to Wordnet: An on-Line Lexical Database. *International Journal of Lexicography* 3.4, pp. 235–244 (cited on p. 26).

Ng, Andrew, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, and Caroline Suen (2015). *Unsupervised Feature Learning and Deep Learning Tutorial*. URL: `http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial` (visited on 11/11/2015) (cited on p. 30).

Och, Franz Josef, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alexander Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine Eng, et al. (2004). A Smorgasbord of Features for Statistical Machine Translation. *Proceedings of the Joint Conference on Humand Language Technologies and the Annual Meeting of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL)*. Boston, Massachusetts, USA, pp. 161–168 (cited on p. 13).

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu (2002). Bleu: A Method for Automatic Evaluation of Machine Translation. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 311–318 (cited on pp. 24, 25).

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). On the Difficulty of Training Recurrent Neural Networks. *Proceedings of the 30th International Conference on Machine Learning*, pp. 1310–1318 (cited on p. 40).

Popović, Maja, Daniel Stein, and Hermann Ney (2006). Statistical Machine Translation of German Compound Words. *Advances in Natural Language Processing*. Vol. 4139. Springer, pp. 616–624 (cited on pp. 13, 63).

Porter, Martin (2001). *Snowball: A Language for Stemming Algorithms* (cited on p. 26).

Rabiner, Lawrence and Biing-Hwang Juang (1986). An Introduction to Hidden Markov Models. *Acoustics, Speech, and Signal Processing Magazine* 3.1, pp. 4–16 (cited on pp. 15, 17).

Rosenblatt, Frank (1962). Principles of Neurodynamics. Spartan Book (cited on p. 30).

Rumelhart, David, Geoffrey Hinton, and Ronald Williams (1986). Learning Representations by Back-Propagating Errors. *Nature* 323.6088, pp. 533–536 (cited on pp. 30, 33, 35).

Russell, Stuart (1995). Artificial Intelligence: A Modern Approach. Prentice-Hall (cited on pp. 30, 31).

Schiller, Anne, Simone Teufel, and Christine Thielen (1995). Guidelines Für Das Tagging Deutscher Textcorpora Mit STTS. *Manuscript, Universities of Stuttgart and Tübingen* 66 (cited on p. 15).

Schmid, Helmut (1994). Probabilistic Part-of-Speech Tagging Using Decision Trees. *Proceedings of the International Conference on New Methods in Language Processing*. Vol. 12. Manchester, UK, pp. 44–49 (cited on p. 64).

– (1995). Improvements in Part-of-Speech Tagging with an Application to German. *Proceedings of the Association for Computational Linguistics (ACL) Special Interest Group for Linguistic Data and Corpus-Based Approaches to Natural Language Processing (EMNLP)* (cited on p. 64).

Schuster, Mike and Kuldip Paliwal (1997). Bidirectional Recurrent Neural Networks. *Transactions on Signal Processing* 45.11, pp. 2673–2681 (cited on p. 38).

Schwenk, Holger (2007). Continuous Space Language Models. *Computer Speech & Language* 21.3, pp. 492–518 (cited on p. 21).

Sennrich, Rico, Barry Haddow, and Alexandra Birch (2015). Neural Machine Translation of Rare Words with Subword Units. *arXiv preprint arXiv:1508.07909* (cited on p. 86).

Shen, Libin, Anoop Sarkar, and Franz Josef Och (2004). Discriminative Reranking for Machine Translation. *Proceedings of the Joint Conference on Humand Language Technologies and the Annual Meeting of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL)*. Boston, Massachusetts, USA, pp. 177–184 (cited on p. 13).

Smith, Aaron, Christian Hardmeier, and Jörg Tiedemann (2014). BLEU is Not the Colour: How Optimising BLEU Reduces Translation Quality (cited on p. 28).

Snover, Matthew, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul (2006). A Study of Translation Edit Rate with Targeted Human Annotation. *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas (AMTA)*. Cambridge, Massachusetts, USA, pp. 223–231 (cited on p. 25).

Sukhbaatar, Sainbayar, Jason Weston, and Rob Fergus (2015). End-to-End Memory Networks. *Advances in Neural Information Processing Systems (NIPS)*, pp. 2431–2439 (cited on p. 54).

Sutskever, Ilya, Oriol Vinyals, and Quoc Le (2014). Sequence to Sequence Learning with Neural Networks. *Advances in Neural Information Processing Systems (NIPS)*, pp. 3104–3112 (cited on pp. 12, 22, 30, 48, 49, 54).

Tjong Kim Sang, Erik and Fien De Meulder (2003). Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. *Proceedings of the 7th Conference on Natural Language Learning (CoNLL)*. Vol. 4. Edmonton, Canada, pp. 142–147 (cited on p. 21).

Toutanova, Kristina, Dan Klein, Christopher Manning, and Yoram Singer (2003). Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (HLT-NAACL)*. Boston, Massachusetts, USA, pp. 173–180 (cited on p. 69).

Toutanova, Kristina and Christopher Manning (2000). Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. *Proceedings of the 2000 Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC)*. Hong Kong, China, pp. 63–70 (cited on p. 69).

Viterbi, Andrew (1967). Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *Transactions on Information Theory* 13.2, pp. 260–269 (cited on pp. 18, 46).

Wallach, Hanna (2004). Conditional Random Fields: An Introduction. *Technical Reports (CIS)*, p. 22 (cited on p. 21).

Welch, Lloyd (2003). Hidden Markov Models and the Baum-Welch Algorithm. *Information Theory Society Newsletter* 53.4, pp. 10–13 (cited on p. 20).

Weller, Marion and Ulrich Heid (2012). Analyzing and Aligning German Compound Nouns. *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC)*. Istanbul, Turkey, pp. 2395–2400 (cited on p. 64).

Widrow, Bernard and Marcian Hoff (1960). Adaptive Switching Circuits. Defense Technical Information Center (cited on p. 30).

Williams, Philip, Rico Sennrich, Maria Nadejde, Mathias Huck, and Philipp Koehn (2015). Edinburgh's Syntax-Based Systems at WMT 2015. *Proceedings of the 10th of the Association for*

Computational Linguistics (ACL) Workshop on Statistical Machine Translation (WMT). Lisbon, Portugal, pp. 199–209 (cited on p. 72).

Xu, Jun and Hang Li (2007). Adarank: A Boosting Algorithm for Information Retrieval. *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics Special Interest Group on Information Retrieval (ACL-SIGIR)*. Prague, Czech Republic, pp. 391–398 (cited on p. 81).

Zhang, Sixin, Anna Choromanska, and Yann LeCun (2015). Deep Learning with Elastic Averaging Sgd. *Advances in Neural Information Processing Systems (NIPS)*, pp. 685–693 (cited on p. 86).

List of Figures

List of Tables