Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

# Project Thesis
# Autumn 2017 Computer Science

## Neural Machine Translation

| | |
|---|---|
| **Authors** | Daniel Einars,<br>Nicolas Hoferer |
| **Referees** | Mark Cieliebak,<br>Kurt Stockinger,<br>Jan Deriu |
| **Date** | January 10, 2018 |

Zurich University
of Applied Sciences

School of
Engineering
InIT Institute of Applied
Information Technology



# DECLARATION OF ORIGINALITY
# Project Thesis at the School of Engineering

By submitting this Project Thesis, the undersigned student confirms that this thesis is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the Project thesis have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date: Signature:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

The original signed and dated document (no copies) must be included after the title sheet in the ZHAW version of all Bachelor thesis submitted.

# Preface

We would like to thank our project advisers Dr. Mark Cieliebak, Dr. Kurt Stockinger and Jan Deriu for their support, advice and guidance. Furthermore we would also like to thank the Institute of Applied Information Technology at the Zurich University of Applied Sciences for providing the necessary computational resources for our experiments.

# Abstract

With the rising availability of computing power it has become possible to develop and train Neural Machine Translation (NMT) models. This, in turn, has led to a rise in approaches to machine automated translation, especially in the past ten years. In order to provide an oversight of the most common NMT-Models, their approaches, performance and tuning capabilities, this thesis aims to complete the following tasks.
This thesis attempts to complete the following tasks:

1. Investigate, implement and measure the performance of current state-of-the-art Neural Machine Translation Models(NMT) and our own NMT Model.
2. Investigate and attempt to provide a detailed explanation of the inner workings of thought-vectors.
3. Investigate the influence of various hyper-parameters used to fine-tune NMT models.
4. Compare the performance of Google's and DeepL's production translation systems.

The results detailed in this thesis can be used as a starting point to build NMT models. The performance evaluations allowed us to define which approaches were successful and which weren't. It also became apparent, that at the bare minimum, the NMT models should produce BLEU of 15 in order to generate understandable translations. After having researched current NMT techniques, such as bidirectional layers, attention, subword encoding, as well as realizing the importance of correct data and preprocessing this data, we have learned that it becomes surprisingly easy to achieve a satisfactory BLEU score. However, completing this task without access to multiple GPUs in order to parallelize the training is still not recommended, as the training time would be too long.

# Contents

# 1. Introduction

The rapid pace at which we have overcome barriers of communication in the past century is astounding. When examining the development of publicly available communication methods, starting with the introduction of traditional carrier mail [1] to the inception of the internet [2], it would appear that one of our only shared goals is the dismantling of communication barriers. The explosive development of the internet and its infrastructure has contributed its fair share to this end and enabled anyone with access to it, which is currently 48% of world's population [3], to connect with millions of others.

However, being able to connect with people from all over the globe does not necessarily imply communication is possible. In 2009 there were approximately 6909 unique languages being spoken [4] and translating accurately between these was, for a long time, either reserved for professional interpreters or involved a combination of tedious dictionary look-ups and an improvised sign-language.

The advent of Google's online Translator Service in 2006 has marked a mile-stone in making translation services available to the public. During its tenure it has continuously improved on it's features; be it the number of supported languages, which grew from two to 103 in the span of ten years, or features such as translating text from images and mapping the translated text back onto the image [5], [6]. However, while the service provided instantaneous translation, in its early days the produced output often fell short in terms of grammatical correctness and cohesiveness. Even though the field has come far since its inception, there are still a number of challenges to overcome.

They are as follows:

1. Ambiguity - Words may contain multiple meanings and depending on their context, only a subset apply [7].

2. Non-Standard Terminology - This refers to the use of language constructs which do not adhere to the official language documentation. A popular example of this are the abbreviations and emojies used in tweets.

3. Named Entities - To a machine, a name appears like any other word. It is a set of characters. However, they are typically not translatable and thus the machine needs to be in a position to accurately identify named entities.

A number of approaches have been developed to tackle these, with varying degrees of success. They can be roughly categorized into the following three classes.

**Static Translation**

The first implementation was a simple word-for-word translation. While this is a relatively simple technique, it can produce some useful results.

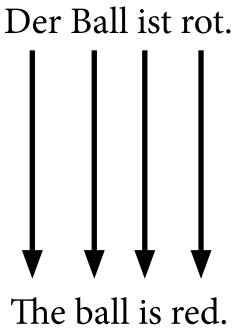


Figure 1.1.: Word-for-Word translation simple example

As Figure 1.1 shows the translation quality is relatively accurate and understandable at a first glance for this specific example. However, this remains true only for sentences with a simple structure.

The translation shown in Figure 1.2 shows that the quality of the static translation approach quickly deteriorates if the source sentence structure becomes slightly more complex.

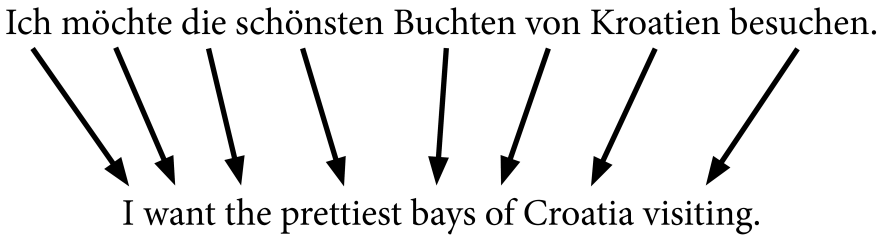


Figure 1.2.: Word-for-Word translation advanced example

**Rule-Based Translation**

The earliest attempt to solve this problem came into existence in 1933 through the work of Petr Troyanskii. Troyanskii devised a set of logical symbols in order to express grammatical relationships across languages. His process involved a human editor preparing the source text by reducing all full forms to their stems and defining the grammatical relationships. The machine would then translate these into the target language and lastly the human editor would convert the produced stems and symbols into the target language [8]. The results of this rule-based approach can be seen in the re-ordering of the produced sentence in Figure 1.3.

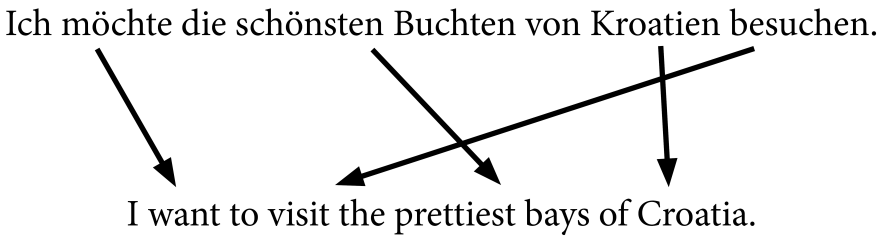


Figure 1.3.: Rule-Based translation example

Variations of this rule-based approach continued to be developed and appeared to solve the problem of grammatical correctness. In essence, linguists would work with programmers to supply the system with rules, thus increasing the quality of the produced output. While this approach does offer an increase in translation quality, it does have one large drawback. Programming all the grammar rules for all languages is a time-consuming task, even for a single language. Implementing these rules for all currently spoken languages is simply not feasible.

**Statistic-Aided Translation**

The next attempt at improving machine translation involved the use of statistical models. This required large data sets of text available in two languages. The proceedings from the European Parliament were often used as a base because these are translated into 21 languages by professional translators, thus ensuring the data's accuracy [9]. The statistic-aided translation works in four steps.

1. Break the input-sentence into individual pieces.
2. Find all translation possibilities of these individual pieces.
3. Generate all combinations of translated pieces.
4. Iterate over statistical models to discover which combination is the most likely correct outcome.

While this does produce reliable results, it too has it's limitations. Because these models rely heavily on the availability of translated texts, translation quality suffers if the parallel corpa aren't available. A solution to this is to use an intermediate language. For example translating from the Romansh to Swedish, both languages with less than ten million native speakers [10], the source is first translated to English and then from English to the target language. It is possible that, during this process, the translation quality declines. While some direct translations for

highly language-specific terms between Romansh and Swedish might exist, they are lost in the process of translating Romansh to English if the English language does not have constructs for these. Furthermore, building and maintaining these models is a highly complex task, requiring experts to continuously fine-tune the statistics.

The rapid advancements in machine learning as well as increase in computational power has enabled computer scientists to apply deep-learning theory to translation and, although different implementations vary widely in terms of accuracy, speed and reliability, it has, in general, shown the most promise of succeeding. This, combined with the fact that translation is a multi-billion dollar market [11], has lead to an increase in attention from the scientific community resulting in a large number of implementations. In order to provide an overview of different methodologies, their success and effectiveness, this project thesis will focus on the following aspects:

1. Build and reproduce the results of selected recently published papers.
2. Build and evaluate a functioning Neural Machine Translation Model.
3. Compare the performance of our model against the performance of the rebuilt models.
4. Explore the possibilities of fine-tuning models and define which attributes increase the translation quality.
5. Compare the performance of Google's and DeepL's production translation systems.

The thesis is structured as follows. Chapter two briefly summarizes work related to this thesis. The fundamental concepts of machine learning, homing in on terminology and theory commonly applied in NMT and common evaluation methods, are

explained in chapter three. Chapter four is dedicated to summarizing the selected published approaches as well as show-casing ours. Chapter five and six details the selected data and experiments. The results are described and discussed in chapter seven.

# 2. Related Works

The first breakthrough in neural machine translation was made by KyungHyun Cho's team in 2014 when they devised a deep-learning system that was able to learn how to translate between two languages without any human intervention [12]. Their approach was to use a parallel corpora to train a neural translation model which, in turn, was implemented as a recurrent neural network (RNN) and incorporated the use of encodings in a architecture referred to as Sequence-to-Sequence [12], [13]. These terms will be discussed in more detail in the chapter *Theoretical Principles*. For now it is important to note that the use of these produced a remarkable increase in accuracy.

While deep-learning was already being used to some degree for translation tasks in applications such as Microsoft's Skype [14], Cho's work revealed how much more efficient these system could become in the future. It essentially laid the groundwork for many following approaches, some with minor adjustments, others with new approaches. The success and impact of Cho's work lead the researches Koehn and Knowles to identify new challenges which Neural Machine Translation had to overcome [15].

1. Loss of quality when translating out side of the domain of trained data.
2. The need for large data sets to ensure quality.
3. Loss of quality when facing low-frequency words.
4. Loss of quality when translating long sentences.
5. The attention-mechanism popularized in Cho's work can lead to divergence.
6. The Beam-Search mechanism only increased translation quality in small search spaces.
7. The difficulty associated with interpreting Neural Translation Models.

The final point mentioned piqued our interest. It was our goal to understand how Neural Machine Translation is able to outperform every other translation approach to date, as well as understand what some of the influencing factors on the translation quality among are.

During our research we came across a vast number of implementations with varying degrees of complexity and it became apparent that, in order for us to understand which factors positively influenced translation quality, we would have to start by implementing simpler models and move forward from there.

# 3. Theoretical Principles

The first part of this chapter is dedicated to explaining fundamental terminology and theoretical concepts in machine learning, followed by concepts specific to NMT. It is important to note that the following definitions and explanations are restricted to information specific to NMT.

## 3.1. Recurrent Neural Networks

In this section we provide a brief summary of Artificial Neural Networks (NNs), sometimes also refereed to as *Feed-Forward Neural Networks*. Our thesis uses an advanced form of *NNs* known as *Recurrent Neural Networks* (RNNs).

**Neural Networks**

The concept of *Neural Networks* (NN) refers to a mathematical model inspired by the structure of the human brain (human neural network) [16]. NNs are commonly composed of an input layer, multiple hidden layers and an output layer [17]. Each layer in a NN can have multiple neurons. Based on the input and its respective weights, the Neuron calculates an output. The function responsible for this calculation within a neuron is referred to as the *Activation Function*.

Figure 3.1 depicts a basic structure of an NN, with a single hidden layer. In this figure the inputs are denoted with *x*, the weights with *w*, the function of the neurons as $\delta$, the activation functions as *a* and the predictions produced by the output layer are denoted with *y*.

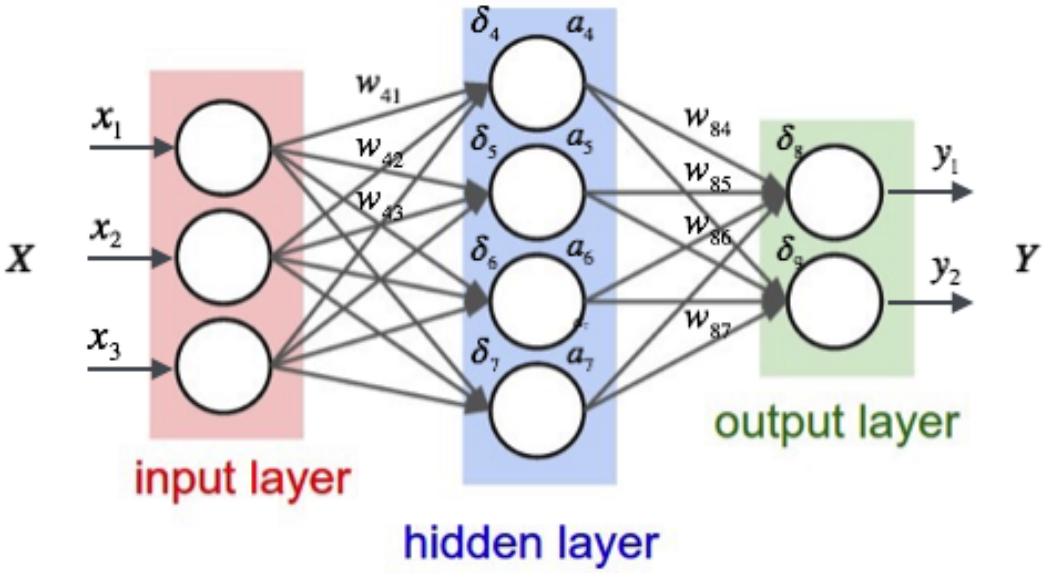


Figure 3.1.: Basic Feedforward Neural Network. [18]

At a basic level, NNs are trained by being provided with training data consisting of two components. One component is used to give the NN data from which it can generate a prediction and another component is used to verify the accuracy of its prediction. The network computes a prediction based on the functions of each neuron, the input weights and their activation function. In order to evaluate the performance a *loss* or *cost* is calculated using a *loss-* or *cost-function* at given intervals. A low loss indicates that the predictions being made by the NN are more often correct than incorrect. While a number of various loss-functions exist, they operate by comparing the predictions with correct results and producing a measure of how far the predictions deviate from the correct results. Weights are continuously optimized during the training process in order to minimize the loss and thus increase prediction accuracy.

For this work we have decided to use a *loss-function* known as *Cross-Entropy*, where $\hat{y} \in \mathbb{R}^{|V|}$, which means that $\hat{y}$ is a probability distribution over the vocabulary. The formula of the *Cross-Entropy* is shown in equation 3.1.

$$J^{(t)}(\Theta) = -\sum_{j=1}^{|V|} y_{t,j} * log\hat{y}_{t,j} \tag{3.1}$$

While we do optimize our network to minimize its loss, we evaluate the performance by calculating a BLEU score, which provides us with a measure of how close a predicted translation (hypothesis) is to the correct translation (reference). While there have been attempts at optimizing neural machine translation models using the BLEU score [19], [20], [21], they have not yet been successful.

**Backpropagation**

Backpropagation is a method used to decrease a neural network's loss by updating weights through the use of algorithms referred to as *optimizers*. Our work utilizes various optimizers which are based the *gradient-decent* optimizer and works as follows:

1. The NN is fed with data, computes predictions and loss.
2. The loss is propagated backwards through the NN, during which each weight is updated based on its influence on the loss.

Weights are adjusted by subtracting a value indicating the weight's influence on the loss from its previous value. The value which indicates a weight's influence is derived by evaluating the partial derivative of the error function with respect to the corresponding weight and then multiplying it with the learning rate, which indicates how much weights are updated.

Equation 3.2 shows the update rule of a weight, where $w_i$ indicates the weight, $\eta$ the learning rate and $\frac{\partial E}{\partial w_i}$ represents the partial derivative of the loss $E$ with respect to the weight $w_i$ of the weight [22].

$$w_i = w_i - \eta * \frac{\partial E}{\partial w_i} \tag{3.2}$$

**Recurrent Neural Networks**

In addition to the information flow, a *Recurrent Neural Network* (RNN) posses a temporal dimension, meaning it is able to take into account the position in time of each input. This is achieved by adding a connection from a neuron's output to its input as shown in Figure 3.2 [23].

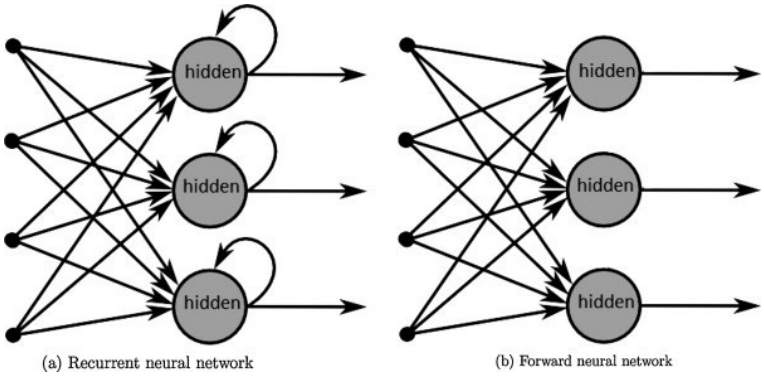


Figure 3.2.: Side by side comparison of a NN's and RNN's structure [24]

.

Figure 3.3 depicts how, in a basic RNN with three cells, the inputs are processed. Cell *A* receives the first input (denoted as $X_{t-1}$) and produces its state (denoted with $h_{t-1}$). When processing the next input $X_t$ it is combined with the state $h_{t-1}$ to compute the next step $h_t$. With an RNN, the first input is passed to the RNN cell, and the next input and the output produced by the previous cell is passed to the next cell. Figure 3.3 depicts a basic RNN containing three cells.

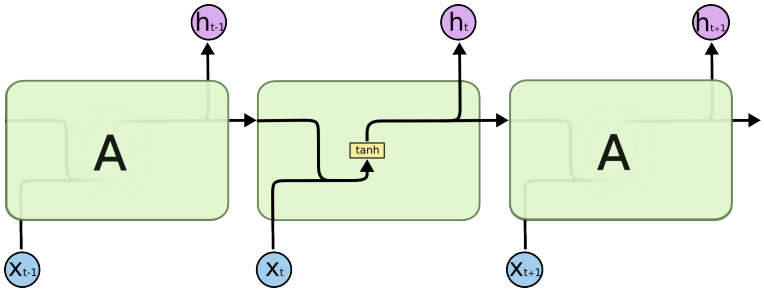


Figure 3.3.: Example of a basic RNN with three cells [25]

.

**Vanishing / Exploding Gradients:**

Due to built-in recurrence in RNNs, problems such as vanishing or exploding gradients can occur [26].

**Exploding Gradients:**

The first problem caused by the RNN's recursive structure are *exploding gradients*. Gradients, which become exceedingly large, are referred to as *exploding gradients* [27]. These can occur during backpropagation where the gradient is a product of multiple jacobian matrices. If the norm of these matrices is large, the gradients have a tendency to become infinite, thus negating a network's ability to learn. A solution this problem, proposed by Y. Bengio, is to *clip* the gradient, should its norm exceed a defined threshold [26].

### Vanishing Gradients

The issue of *vanishing gradients* can occur during the training period while minimizing the loss with respect to the weights. In order to calculate the gradient we take the derivative of the loss function with respect to the weights. During this process the gradient flows backwards through the RNN and the respective activation functions. If activation functions such as *tanh*, which produce results in the domain of [-1, 1], the repeated differentiation by application of the *chain-rule* causes the result to consistently move closer to zero, resulting in a vanishing gradient [27]. A gradient of, or close to, zero leads to only only minuscule updates to the weights in this network, which leads to a plateauing of the network's learning rate. In essence, it ceases to learn. However, this can be avoided by using *Long Short-Term Memory* (LSTM), which is an advanced type of RNN.

### LSTM

The introduction of *LSTMs* by Schmidhuber and Hochreiter tackles the problem of vanishing gradients and introduces the capability to store long-term information [28]. The main idea behind LSTMs is the cell state, which represents the information stored in a LSTM cell in a high dimensional vector. The LSTM cell contains three types of gates, which operate on the cell state.

**Forget gate:** This gate is responsible for deciding which information to keep and which to forget. It does this through the use of a sigmoid layer, which produces an output in the domain of [0, 1] for each number in the cell state. An output of one results in information being kept while an output of zero results in a complete removal of information. Equation 3.3 shows the formula of the forget gate.

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \tag{3.3}$$

**Input gate:** The input gate decides which new information should be stored in the cell state. In order to store new information, a sigmoid layer initially decides which values from the cell state should be updated, denoted as $i_t$ in equation 3.4, in an identical process to the sigmoid layer in the forget gate.

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{3.4}$$

After having identified which values to update, a tanh layer computes which input-values to store in the cell state. This value is denoted as $C_t$, the formula is shown in equation 3.5.

$$\tilde{C}_t = \tanh(W_c * [h_{t-1}, x_t] + b_C) \tag{3.5}$$

Following this, the cell state is updated by multiplying the old state by $f_t$, to forget what the forget gate decides, after which we take the sum of $i_t * C_t$ and the modified state. This is showed in equation 3.6.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.6}$$

**Output gate:** The output gate initially computes which values from the cell state to include in the output with an sigmoid layer. Then it brings the values from the cell state into the domain of [-1, 1] with a tanh layer and, based on the output from the previous sigmoid layer before, the output get generated.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3.7}$$

$$h_t = o_t * \tanh(C_t) \tag{3.8}$$

Figure 3.4 depicts a basic structure of a RNN, similar to Figure 3.3, however, in this scenario the inner workings highlight the use of *LSTMs* in an individual cell.

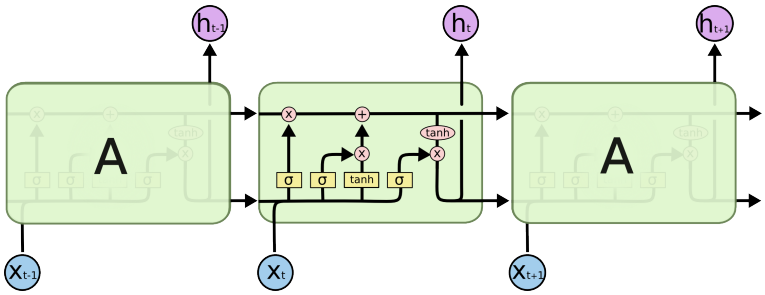


Figure 3.4.: Example of a LSTM with three cells [25]

.

## 3.2. Sequence-to-Sequence

The *RNN* and *LSTM* cells are key components of *Sequence-to-Sequence* (seq2seq) models. While they find their primary application in the NMT domain, they can generally learn to map input to output sequences of differentiating lengths. However, while this ability makes them especially favourable in the NMT-Domain, as translated texts do not always have the same number of words in both languages, it also increases their complexity. A Sqe2Seq architecture consists of two recurrent models, a *encoder* and a *decoder*, which are jointly trained.
The two models are trained jointly, thus enabling the system to maximize the conditional probability of the target sequence for a given source sentence. In summary, the encoder's task is to map the source sentence to a fixed-length vector, commonly referred to as the *hidden state*. It sequentially receives the input in the form of *tokens*, which can either be individual words or characters, depending on the specific sqe2sqe implementation. When the first token of a new source sentence is passed to the encoder, a defined *start of sequence symbol* (SOS) is placed at the beginning of the sequence. With each new token being fed to the encoder its hidden state is updated to accommodate the new information. Once the last token has been received a *end of sequence symbol* (EOS) is placed at the end of the sequence. The SOS and EOS serve to inform the decoder where sequences start and end. Once a sequence has been encoded, the entire hidden state of the encoder, also known as a *thought vector*, is passed to the decoder, which uses it as its initial hidden state and begins decoding. It does this sequentially on a per-token basis and feeds the output to a softmax classification layer.

This layer produces a probability distribution, which essentially computes the highest probability of the next token in a sequence and finally produces the translated text. Figure 3.5 depicts a general encoder-decoder architecture, without a SOS-Token.

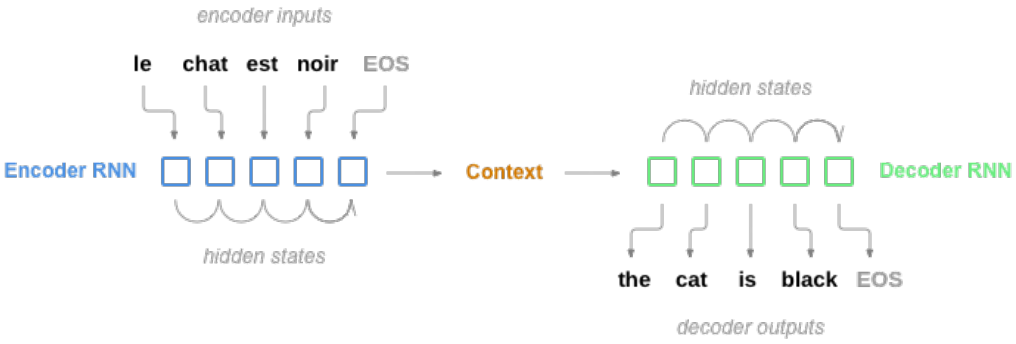


Figure 3.5.: General Encoder-Decoder architecture [29]

.

## 3.3. Attention

The attention mechanism was introduced by Dzmitry Bahdanau, et al. [30]. The idea is to provide a direct short cut between target and source sentences by scoring each word in the source sentence on its importance for predicting the current token. The intuition behind it is to imitate human translation. During human translation the source sentence is continuously evaluated in order to evaluate weather the current translation accurately maps the source. The attention mechanism attempts to solve the problem of having thought vector of a fixed length as the sole representation of the input sequence. Without the attention mechanism, long translations becoming increasingly poor because the entire sequence is compressed in a single thought vector, which is updated after every encoding step. However, attention not only improves the ability to decode tokens from a thought vector, it also helps to improve *alignment*. In the context of NMT, align-

ment referrers to the process of defining which part of a input sequence is relevant to each word of the output sequence. What follows is a brief explanation of the attention mechanism.

**Encoding:** In a vanilla encoder-decoder model, the encoder encodes the input into a single vector of fixed length. With the attention model, the model will require the output from the encoder after each input step. In the original paper these states are referred to as ”annotations” [30].

**Decoding:** During each time step of encoding, multiple steps are executed to apply the attention mechanism. These steps are explained in detail below.

1. **Alignment:** During this step each *annotation* receives a score which expresses the degree to which the current annotation maps to the current output.

2. **Normalizing Alignment Scores:** In order to compute a probability distribution, which shows the probability of each annotation with respect to its current time step, the alignment scores are normalized through the use of a softmax function.

3. **Producing a new thought vector:** A new thought vector is produced by multiplying each annotation by its corresponding weight.

4. **Decoding the next output:** Instead of using the initial thought vector as is the case with a vanilla RNN, the updated thought vector is used.

There are various other attention techniques such as Luong’s *Scaled-Luong* [31], which was inspired by normalized form of Bahdanau attention mechanism, and Bahdanau’s *Normed Bahdanau* [30], which, in turn, was inspired by the Saliman’s weight normalization [32].

One potential problem associated with the attention mechanism is its increase in computational cost. For example when the source and target sequences exceed a length of 50 tokens, a total of $50 * 50 = 2500$ attention values need to be computed. Figure 3.6 shows an example alignment matrix.

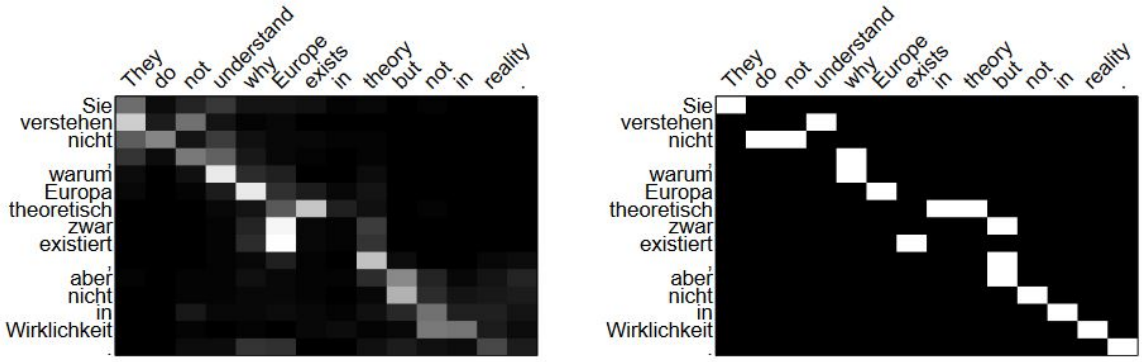


Figure 3.6.: Visualization of the attention alignments.(Left attention alignments of a NMT system, Right gold attention alignments) [31]

## 3.4. Beam Search

The decoder explanation highlighted its greedy approach, during which a the next translated word is selected based on the highest local probability. Due to this, sentences such as *"Das Wettter ist heute wirklich sehr schön!* are translated to *"The weather today is nice!"*, where as the correct translation would be closer to *"The weather today is very nice!"*. Figure 3.7 depicts a greedy decoder, which selects the next word based on the its local probability.

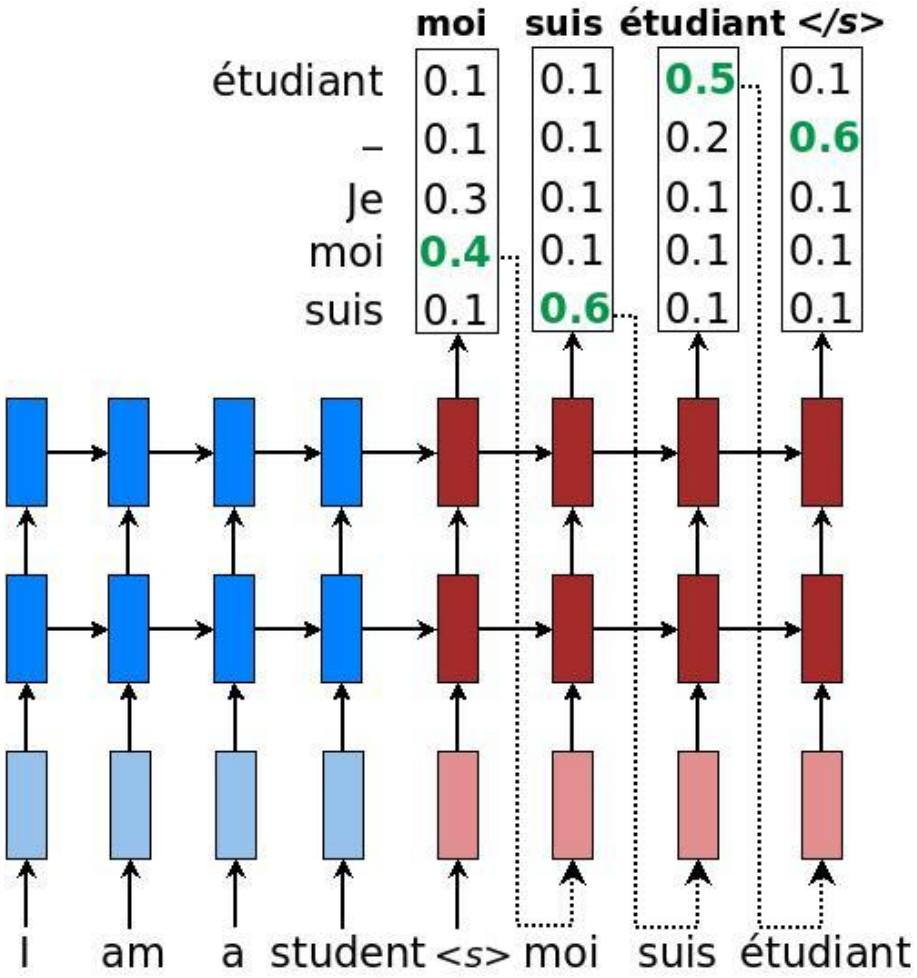


Figure 3.7.: Example of a greedy decoder, where at each time step the word with the highest probability is selected. [33]

In order to mitigate this it is necessary to optimize the translations for their entire probability. This is done by applying a technique known as *beam search*. The beam search considers the *n* best hypotheses at every time step, where the *n* is

referred to as *beam-width*. A larger beam-width results in an increase in translation accuracy, however, at the cost of decoding time.

## 3.5. Teacher-Forcing

Teacher-Forcing is a mechanic aimed at improving translation quality. Specifically, the method which is used to feed the decoder during training changes. The decoder receives the encoder's though vector and predicts the first token at the next time step. Without teacher-forcing, the decoder then receives the output of the previous time step as the input during training and the model learns to predict the next word based on the previous output, as shown in Figure 3.8.

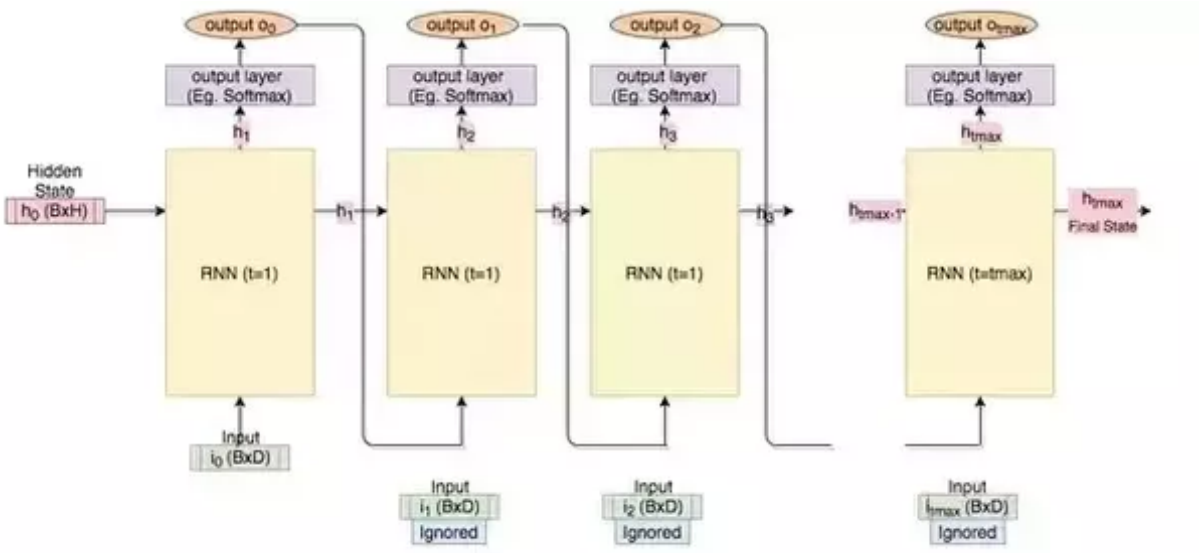


Figure 3.8.: RNN without teacher-forcing [34].

Teacher-Forcing differs at this point by feeding the correct translated word, instead of the prediction, as the decoder's input at each time step. Through this the decoder learns to predict the next word based on the correct translation of the previous time step.

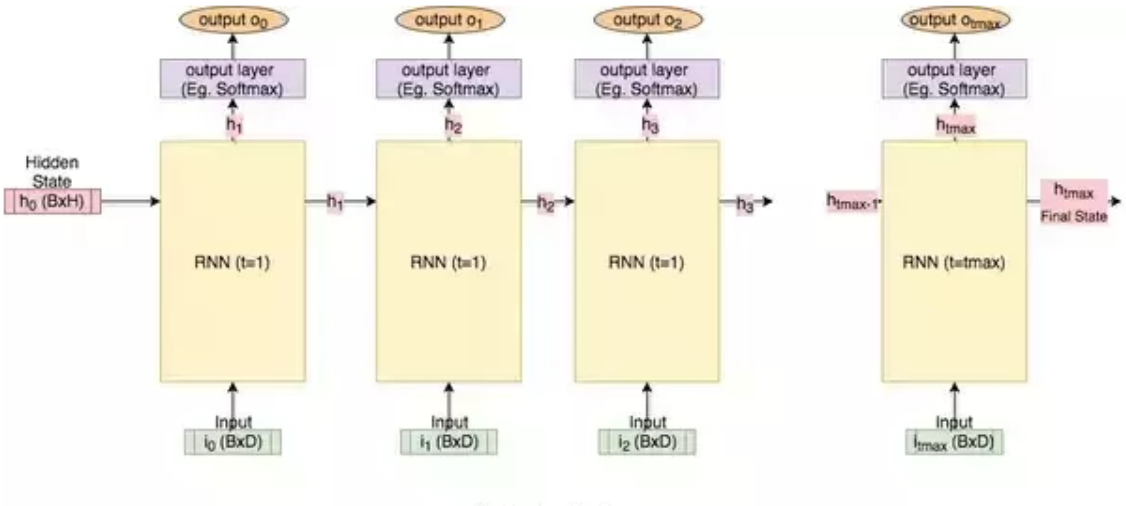


Figure 3.9.: RNN with teacher-forcing [34]

.

## 3.6. Performance Metric

Through the rise in the sheer variety of NMT Models and architectures, the need for a universal performance metric has arisen. Their inherent complexity make direct comparisons a difficult task. Furthermore, the performance metric must enable their developers to identify which changes result in an increase in translation accuracy. Further more, the performance metric must produce results in a timely manner on large data sets. Due to this constraint, as well as the fact that different human translators could interpret an source sentence differently and thus arrive at different target translations, manual evaluation becomes infeasible and impractical. The following chapter is dedicated to briefly explaining the BLEU, which is perhaps the most commonly method to evaluate performance of NMT Models.

While previously highlighted performance metrics, such as *accuracy*, do provide a general insight into the performance of a standard machine learning models, they are inherently impractical when applied to NMT.

**Accuracy:** In the context of machine learning, *accuracy* refers to the percentage of correct predictions. The use of *accuracy* as a performance indicator is typically employed for classification models. When the model is required to predict whether the subject of an image is a dog or a cat, a simple *true* or *false* suffices. However, in the context of translation, usually multiple translation exist for a source sentence. This means that, while the predicted translation *"I would have rather been outside today."* of the source sentence *"Ich wäre lieber heute an der frischen luft gewesen."* could be marked as wrong because the provided reference translation is *"I'd have rather enjoyed some fresh air today."*, even though the predicted translation is one correct variation.

**BLEU-Score:** The BLEU is a common performance metric used to measure how accurate a predicted target sentence is when compared to its reference. In simple terms, BLEU evaluates a Hypothesis by comparing it to a provided reference and evaluating the presence or absence of words in the hypothesis, as well as the ordering and to which degree they are separated from each other. A BLEU of zero marks a completely inaccurate translation and a score of one marks a flawless translation. The formula for BLEU in is shown in equation 3.9, where $P$ denotes the score, $m_{max}$ the number of times a individual word from the hypothesis found in the reference and $w_t$ the total number of words in the hypothesis.

$$P = \frac{m_{max}}{w_t} \tag{3.9}$$

In practice this means that for the hypothesis *"The bird is in the tree"*, the word *"the"* has a $m_{max}$ value of 2 because it appears twice in the reference *"The bird is in the big tree"* and $w_t$ has a value of 7. Lastly, the individual BLEU-Scores are added up to produce a value between zero and one. In this case the score is 0.85 because the hypothesis is missing the word *"big"*. However, this process does not yet evaluate the position of individual words in the reference. This is done by incorporating a *n-gram precision*. In addition to the above, the *n-gram precision* groups words together, typically into groups of one to four, and repeats the above process. For a *2-gram precision* it takes the first and second word (*"the bird"*) and measures how often this sequence appears in the reference and finally adding the individual BLEU-Scores up.

This method allows to evaluate translations of single sequences, however, in practice we wish to compute the BLEU-Score of an entire corpus. While it is possible to simply calculate the average of each individual BLEU-Score, this method does not provide an accurate evaluation due to the fact that the BLEU-Score tends to reward shorter sequences with higher scores, even when evaluating with *n-grams*. An example of this would be the hypothesis sequence *"the car"*, for which the BLEU-Score would be

$$P = \frac{1}{2} + \frac{1}{2} = 1$$

where 0.5 is awarded for each appearance of the word *"the"* and *"car"* and the *n-gram* score would be 1 as the *"the car"* appears in the hypothesis and reference. In comparison, longer sentences might posses multiple translations For example the source sentence *"I've had a wonderful time at my friend's party, however, I would have wished that her cat hadn't jumped all over my coat"* offers more room for interpretation resulting in hypotheses such as *"Die Geburtstagsparty meiner Freundin war sehr schön, jedoch hätte mich mir gewünscht dass sich ihre Katze auf meinen Mantel gelegt hätte"* or *"Ich hatte eine wunderschöne Zeit an der Geburt-*

*stagsfeier meiner freundin, jedoch hätte ich mir gewünscht dass ihre Katze nicht auf meinen Mantel gesprungen wäre"*. This can be mitigated by providing multiple references for each hypothesis, however, in practice, we had difficulty to obtain parallel corpses which provide these. In order to mitigate this shortcoming a *brevity-penalty* is applied. If the total length of the reference corpus is larger or equal to the total length of the source corpus, the penalty defined as $e^{1-r/c}$, where $r$ defines the length of the reference corpus and $c$ the length of the source corpus. The second drawback is BLEU's inability to evaluate languages which lack word-boundaries [35].

The major advantage of being able to compute the BLEU score in an automated fashion, means that it allows us to directly compare different approaches, even if they are not trained and tested on the same data-set or the same languages. For this reason we chose this performance metric as our primary performance indicator.

## 3.7. Vocabulary

The vocabulary in NMT refers to a specific set of words, which are recognized by the encoder and decoder and, thus, correctly encoded or decoded. However, due to limited availability of memory and computational resources, the vocabulary also has to be limited to a fixed size ($k$). Having a language's entire vocabulary in memory is not a feasible approach. The English language for example contains approximately 1 Million words [36], which is already a large set to keep in memory. Additionally, some languages, such as German, allow individual words to be joined to form new words. For example the German word *"Synapse"* (a *synapse* in English) and *"Zerrung"* (*tearing a muscle due to overstraining* in English) can be joined to form the word *"Synapsenzerrung"*, which, loosely translated, would

refer to the act of tearing a synapse and being in a state of mental exhaustion accompanied by headache. In order to limit memory use, the vocabulary is limited to the 40'000 most frequent words in the training set. However, this would severely limit the vocabulary for languages, which allow for word-joining. To mitigate this, a number of techniques such as *Hybrid Word-Character Models* [31] or *Subword Encoding* [37] exist to increase the vocabulary size without the need for increased memory resources. In our thesis we decided to opt for the use of optimizing the vocabularies with the *Subword-Encoding* approach and will briefly explain it.

## 3.8. Subword Units

The idea behind subword units is to divide words into a limited set of common pieces (subwords). Instead of having to keep entire words in the vocabulary, only the subwords are stored and both the encoder and decoder are able to recognize entire words based on the joined subwords, even unknown words.

One algorithm to divide words into sub-word units is *Byte-Pair-Encoding* [38]. Originally, BPE was an iterative algorithm for data compression. The idea of the original BPE is to learn a codebook for compression by allocating codes to common sequences. For example, in the sequence *aaabdaaabac* the most common byte-pair is *aa*, which can be encoded using the a byte that does not occur in the original sequence, *Z* in this example. The new sequence reads *ZabdZabac*. The next most frequently occuring byte-pair is *ab*, which can be encoded into *X*, forming the new sequence ZYdZYac. The last step encodes *ZY* into *X*, resulting in the encoded sequence *XdXac*, with a character count of 5, whereas the original sequence contained 12 characters. In the context of NMT, BPE works slightly differently.

Table 3.8 shows how different forms of the word *legalize* are split up, where the encoded token is *leg*@@ and the @@ indicate that the word continues. Instead of having to keep all of these words separately in memory, through the use of BPE an open vocabulary can be achieved. This type of encoding can be further applied if sequences such as *alize*, for example, are also encoded. This would allow the decoder to recognize any word ending in *alize*, such as *realize*, without having to know the specific word.

| **Word** | **BPE** |
|---|---|
| legalization | leg@@ alization |
| legalize | leg@@ alize |
| legalized | leg@@ alized |
| legalises | leg@@ alises |
| legalizing | leg@@ alizing |

Table 3.1.: Example of BPE [39]

## 3.9. Dimensionality Reduction

One of our goals was to analyse and understand how the encoder and decoder represents its knowledge. In order to achieve this we decided to plot the relevant thought vectors, however, as thought vectors often exceed three dimensions, making it difficult to distil information from a visual representation, we decided to reduce their dimensions. Two well known algorithms which achieve this task are *principal component analysis* (PCA) and *t-distributed stochastic neighbour embedding* (t-SNE).

PCS's approach for dimensionality reduction without the loss of information is to use the correlation between dimensions to identify the minimum number of variables required to retain the maximum amount of information about the distribution

of the original vector. It accomplishes though the use of *eigenvalues* and *eigenvectors* of the covariance matrix, which have a tendency to follow major directions of variation in the data [40].

While PCA is a purely mathematical algorithm, t-SNE employs probability to achieve its goal. The paper introducing t-SNE states *"t-Distributed stochastic neighbor embedding (t-SNE) minimizes the divergence between two distributions: a distribution that measures pairwise similarities of the input objects and a distribution that measures pairwise similarities of the corresponding low-dimensional points in the embedding."* [41] Essentially it evaluates the originally entered data and aims to find the optimal way to display this in lower dimensions by matching the previously named distributions. While it produces results of higher accuracy when fed with high dimension thought-vectors in comparison to PCA, its methodology is computationally intensive [40]. Unfortunately this limits its area of application precisely where it is supposed to be applied.

Due to the t-SNE's computational requirements, as well as less familiarity than with PCA, this thesis will employ the latter for dimensionality reduction.

# 4. Approaches

This thesis aims to reproduce the performance of three NMT Models by implementing them according to their specifications, as well as evaluate the performance of DeepL's and Google's production systems. In a second step, based on the knowledge gained by implementing *KerasTut*, *FastAI* and *GNMT*, we built our own model, which we named *ZSOE*. Lastly we compare the performance of our model with the two production models and the three rebuilt models. The purpose of this chapter is to discuss the various approaches used to build the below listed models and provide insight into our own model.

1. Keras' Character-Based Model (*KerasTut*)
2. Google's Word-Based Model (*GNMT*)
3. TensorFlow-Talks Model Word-Based (*FastAI*)
4. DeepL's production system (*DeepLProd*)
5. Google's production system (*GoogProd*)
6. ZHAW School of Engineering Thesis Model (*ZSOE*)

It is important to note that we did not approach DeepL or Google for direct access to their production systems, as we found it highly unlikely they would share this proprietary technology. However, we did evaluate their systems' performances by querying the freely available online services, provided by Google and DeepL respectively, and computed the BLEU scores based on produced responses.

## 4.1. KerasTut

The KerasTut's approach uses character-level tokenization in its data preprocessing and employs the previously mentioned teacher-forcing technique. The article

describing the model defined a batch size of 64, one encoder layer and one decoder layer, which are both configured with a latent dimension of 256. The prediction mechanism for the *n+1's* token is implemented by a dense layer after the recurrent layer in the decoder, which applies a softmax evaluation. The size of this dense layer is identical to the size of the vocabulary. The article specified *rmsprop* with categorical cross entropy loss as the optimizer, however, we opted to rely on *adam* as the optimizer with a learning rate of 0.001.

Perhaps the biggest gripe we had with this approach is the lack of a scientific performance evaluation, such as BLEU. Furthermore, their datasets, which were rather small to begin with, were not split into three separate sets (training-, validation- and test-set) making it somewhat difficult to discern if over-fitting had occurred during training. Lastly, advanced techniques, such as *attention*, were not implemented.

However, as this model was intended as an introductory tutorial provided by *Keras*, there was no need for an entirely scientific approach from their perspective. Regardless, we would have found it helpful had the datasets at the very least been split up into three separate ones as this is common practice in today and omitting this, especially in an introductory tutorial, made it cumbersome to distil what was happening behind the scenes. We have used this implementation for learning *Keras* only.

## 4.2. FastAI

FastAI's implementation originates from a TensorflowTalk in Singapore [42]. The model is word-based, relies on pre-trained word embeddings and is configured with a batch size of 128. Its architecture employs two bi-directional LSTM layers as the encoder and one LSTM layer at the decoder, each with a latent dimension

of 128. This means that the encoder is fed an input sequence in its original order and in a reversed order. In theory, this is done to increase translation accuracy of long sequences. Previously the translation accuracy declined as length of the sequence increased due to the thought-vector's representation being more accurate for the end of the sequence, as it had entered most recently. This sometimes resulted a disconnect in rationale between two halves of long sequences. The model uses the *adam* optimizer with an initial learning rate of 0.001 and categorical crossentropy as the loss.

The prediction works by passing the last state vector of the decoder to two dense layers followed by a *softmax* application resulting in the next token. The first dense layer has a latent dimension equal to the dimension of the target language vector. The second dense layer expands this size to reach the size of the target vocabulary size.

While they split their datasets into training, validation and test sets, unfortunately they do not provide performance evaluation metrics. Secondly, the datasets used in the training were not explicitly defined by them. After scouring the relevant domains for the data set we found the one which we believe was used for the training.

## 4.3. GNMT

This model is Google's word-based seq2seq implementation [43], [44]. The model's architecture consists of a two-layer Google neural machine translation encoder and a two-layer LSTM decoder. The Google neural machine translation encoder architecture is a new architecture, which employs residual connections and a modified attention mechanism. Employing residual connections results in the input of a layer being fed element-wise to the output before it is passed on to the next

layer. They use a normed bahdanau attention mechanism but modified it to use the gnmt_v2 architecture. The gnmt_v2 is their novel architecture, where they decrease training time through increased parallelism by modifying their attention mechanism to add a new connection from the bottom layer of the decoder to the top layer of the encoder. They also use a technique known as *dropout* after each recurrent layer to address the issue of overfitting. The *dropout* mechanism randomly drops cells and their connections during the training, thus preventing them from co-adapting thereby decrease the likelihood of overfitting occurring [45]. Exploding gradients are avoided by clipping them at the the global norm. Additionally, their vocabulary is also able to handle words occurring with a low frequency by learning the previously mentioned sub-word units in a state-of-the-art way. Beam Search is employed by their decoding mechanism. The penalty to the required decoding time through this is mitigated by the use of an additional length-normalization technique and coverage penalty. Their training employs the SDG optimizer with a decaying learning rate. Lastly, their model attempts to directly optimize using BLEU instead of a common loss by integrating it with a reinforcement learning approach.

## 4.4. GoogProd

As previously mentioned, we did not reach out to Google enquire about the inner workings of their current production system. However, we did compute its BLUE Score by querying Google Tables, which is able to provide translations.

## 4.5. DeepLProd

DeepL came into the spotlight on August 29th, 2017 by claiming to achieve a BLUE score of 44.7 for English-French translations [46]. This caught our attention

and we therefore wanted to include it in our thesis. To use the production system of DeepL we queried their service through a python library called *pydeepl*, which essentially serves as a wrapper around DeepL's API, and calculated the resultant BLUE score in order to verify their claim.

## 4.6. ZSOE

We build our own word-based seq2seq model in keras. In this paper we refer to this model with "Our Model".
Our model is a word-based seq2seq approach. We decided to implement this model using *Keras's* high-level neural network API [47]. *Keras* is geared towards enabling quick development of neural networks without getting bogged down in the details of TensorFlow by abstracting the implementation. This allowed us to start developing models and experimenting with various approaches.

Our basic model is word-based and does not employ teacher-forcing. Its batch size is 128 and it uses one LSTM layer in the encoder and another in the decoder. A dropout mechanism is applied to each LSTM layer with a keep rate of 0.8, which corresponds to keep 80 % of the cells. Each LSTM layer has a latent dimension of 128. After the last LSTM layer a dense layer of the size of the target vocabulary is applied followed by a softmax to predict the next token. We use the adam optimizer with an initial learning rate of 0.001 and categorical cross entropy as our loss. This model does not use any advanced technique like *bidirectional layers* or some kind of *sub-word encoding*. We decided to implement a simple model to learn how to build models in *keras* and how *seq2seq* works.

# 5. Data

This chapter will highlight the dataset which were used in this thesis. It will detail their origin and structure as well as explain which preprocessing techniques were applied to them in order to prepare them for training. Table 5 lists the four different datasets, each being a parallel corpus containing either German-English or English-French, which we used for our training. For our validation and test datasets we opted to choose the newstest2013 and newstest2015 datasets respectively. They are still mentioned in the individual dataset descriptions as we found it helpful to keep them in mind when comparing their vocabulary occurrences.

| **Name** | **Sequences** | **Languages** |
|---|---|---|
| Tatoeba | 153'868 | English-German |
| FastAIDataset | 18'315 | English-French |
| WMT16 | 4'500'966 | English-German |
| WMT16 1 Mio. | 1'000'000 | English-German |

Table 5.1.: Dataset Summary

The vocabulary for all datasets consists of the 40'000 most common words found in each respective language set. Due to their differences in size, complexity and language, the datasets are not comparable to each other.

## 5.1. Tatoeba

The Tatoeba was discovered during our initial research and accompanied the KerasTut model. The language is the dataset is kept very simple in comparison to the other datasets. It contains both languages in one file, where paragraphs of

the two languages are separated by a tab. Table 5.1 lists the datasets characteristics. *Tar.* and *Src.* refer to the target and source language respectively, while *Trn.*, *Val.* and *Tes.* refers to the split into the respective training-, validation and test set. *Average Length* (Avg. Length) and *Maximum Length* (Max. Length)refer to the average and maximum length of the sequences. It is important to note that a sequence can consist of a single word or multiple sentences. The minimum length was omitted from this table across all sets because it is always one. This results from sequences such as *"hello"*.

| Set | Unique Tokens | Paragraphs | Avg. Length | Max. Length | Not in Vocab. |
|---|---|---|---|---|---|
| Trn. Src. | 13'186 | 153'868 | 7.9735 | 27 | - |
| Trn. Tar. | 26'924 | 153'868 | 7.9147 | 33 | - |
| Val. Src. | 8'780 | 2'998 | 21.6698 | 27 | 3'649 |
| Val. Tar. | 11'982 | 2'998 | 21.2592 | 30 | 5'917 |
| Tes. Src. | 7'359 | 2'170 | 21.7544 | 22 | 2'981 |
| Tes. Tar. | 9'549 | 2'170 | 20.5733 | 32 | 4'719 |

Table 5.2.: Dataset Tatoeba

The vocabulary was constructed of the 40'000 most common words in the respective source and target set. However, this still left some words which appeared with lower frequencies excluded from the validation- and test-set. In the German validation-target and test -target set this amounted to 3'565 and 3'110 tokens respectively while the English equivalents are substantially less with 1'101 and 1'116 occurrences respectively. Figures 5.1, 5.2, 5.3 and 5.4 provide an overview of the tokens with the highest occurrence counts in the validation and test sets of the respective source and target languages, which were not included in the vocabulary and thus marked as *unknown tokens* during the learning sequences.

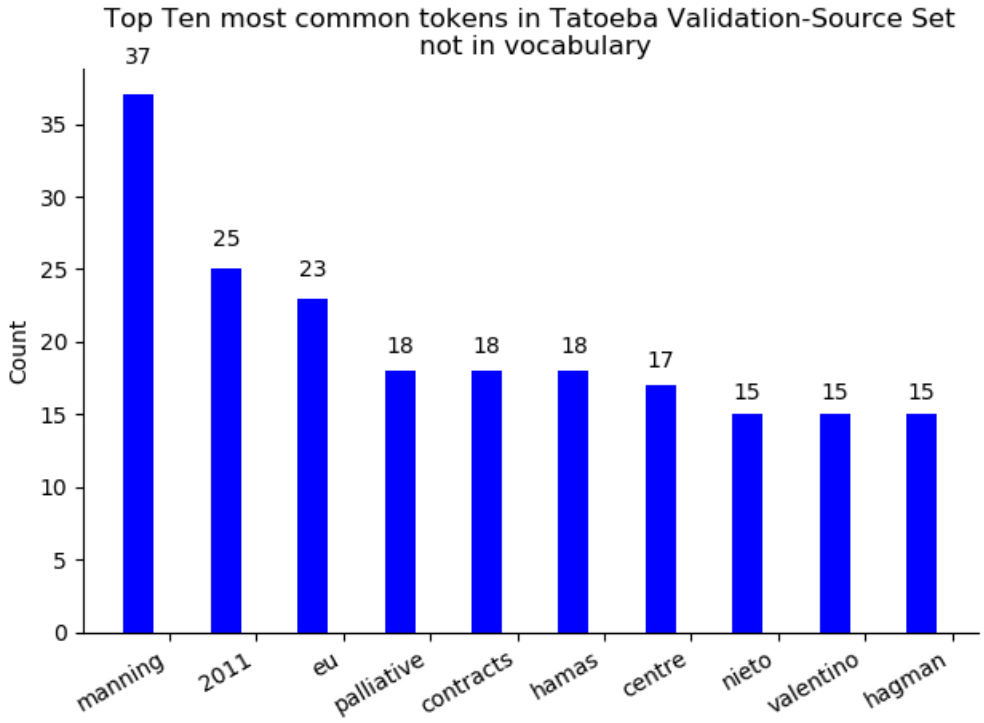


Figure 5.1.: Occurrences of tokens in the Tatoeba Validation-Source Set which were not included in the vocabulary

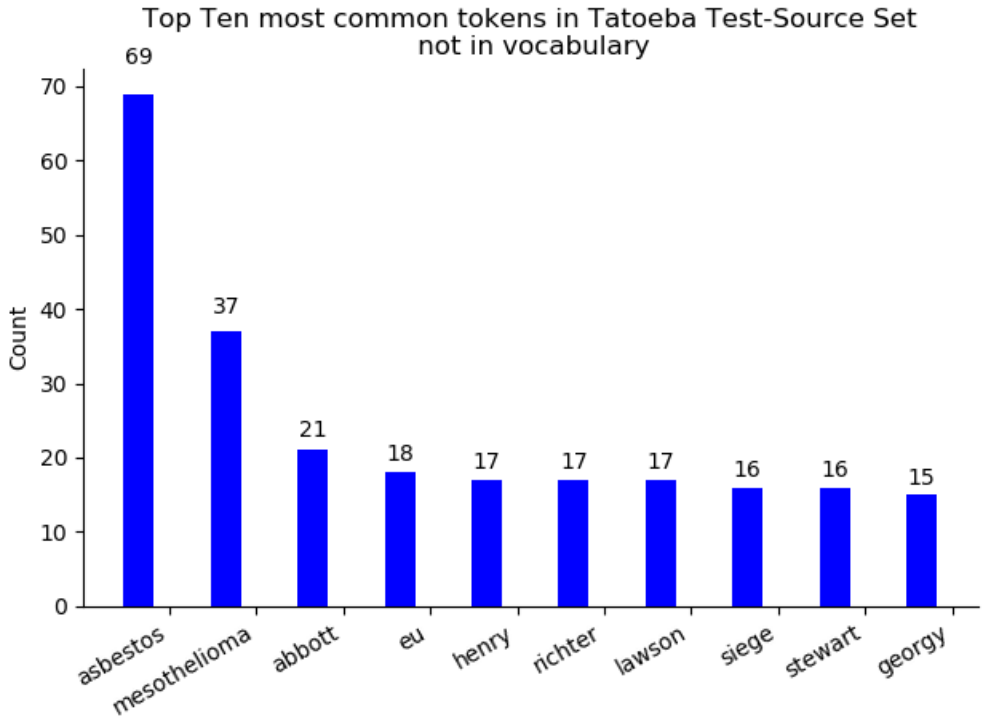


Figure 5.3.: Occurrences of tokens in the Tatoeba Test-Source Set which were not included in the vocabulary

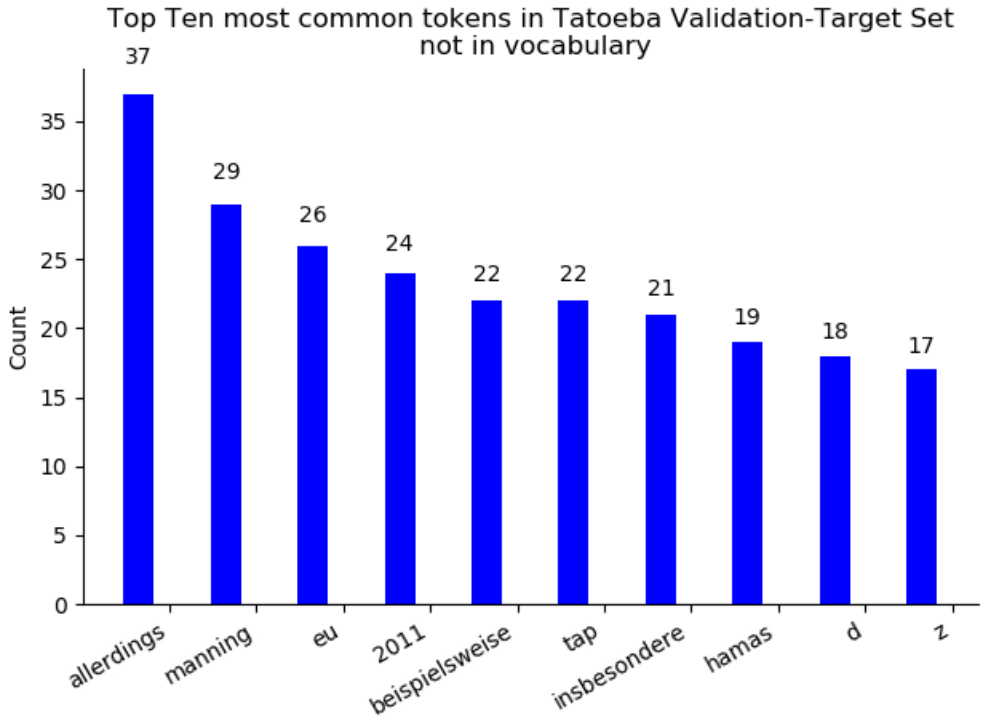


Figure 5.2.: Occurances of tokes in the Tatoeba Validation-Target Set which were not included in the vocabulary

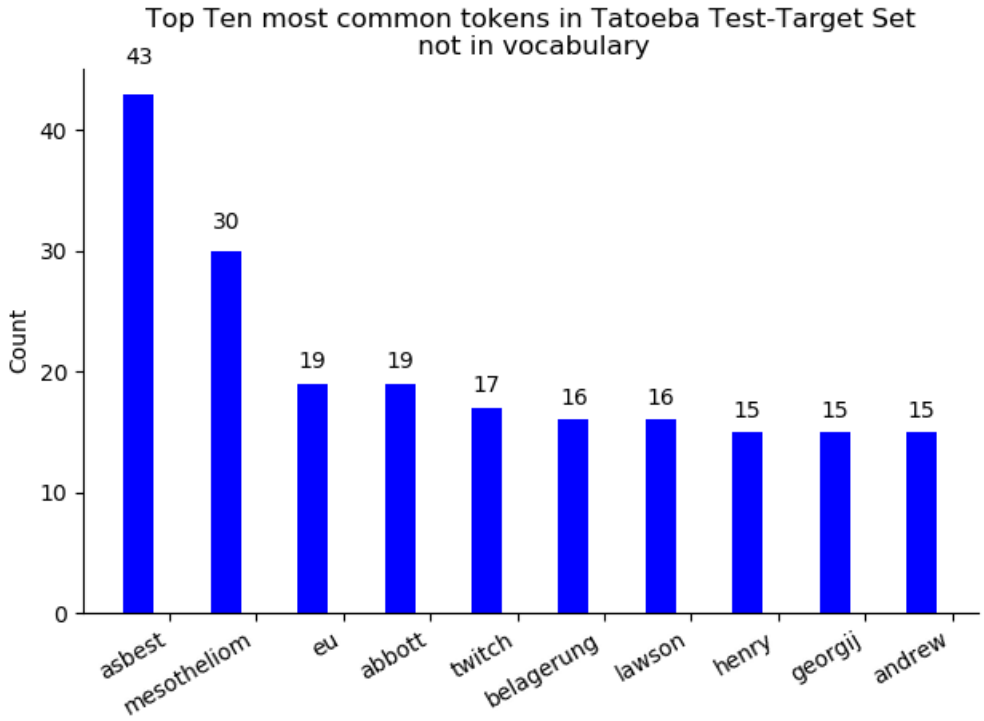


Figure 5.4.: Occurrences of tokens in the Tatoeba Test-Target Set which were not included in the vocabulary

Figures 5.5 through 5.10 give insight into the most common tokens in the respective datasets. Please note that we have removed punctuations from this list as it would completely dominate every figure.

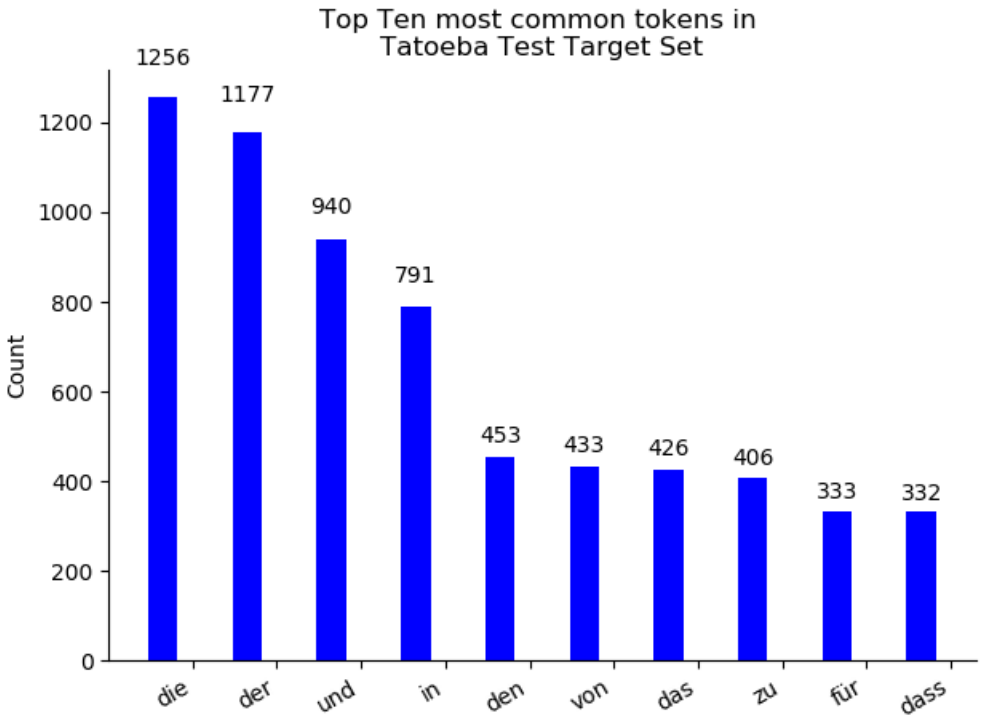


Figure 5.5.: Top Ten tokens in the Tatoeba Test-Target dataset

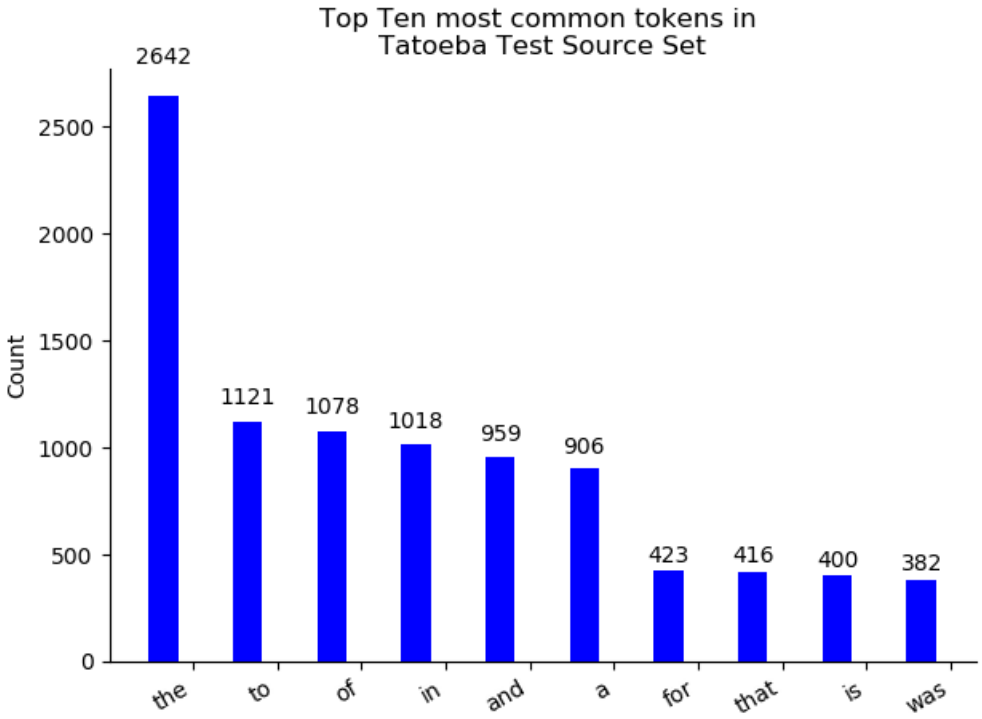


Figure 5.6.: Top Ten tokens in the Tatoeba Test-Source dataset

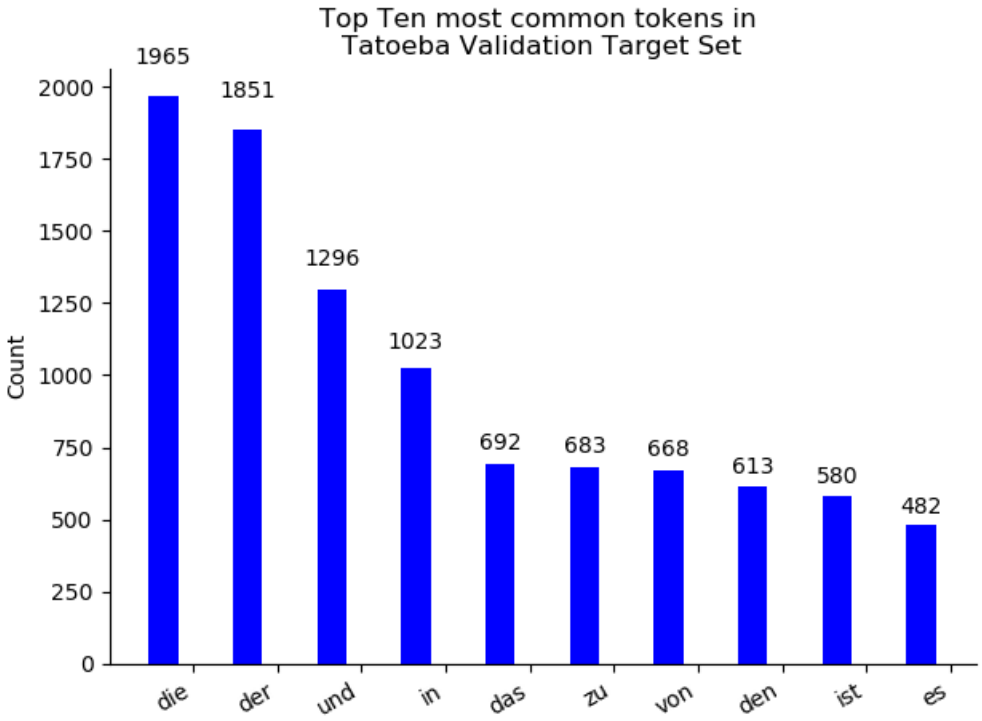


Figure 5.7.: Top Ten tokens in the Tatoeba Validation-Target dataset

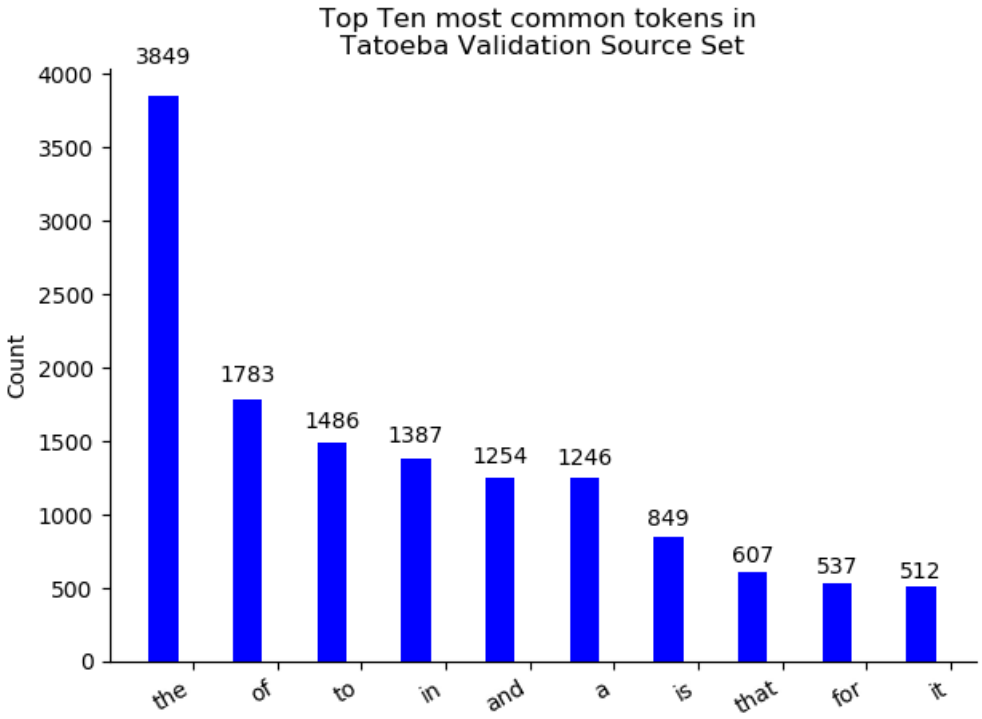


Figure 5.8.: Top Ten tokens in the Tatoeba Validation-Source dataset

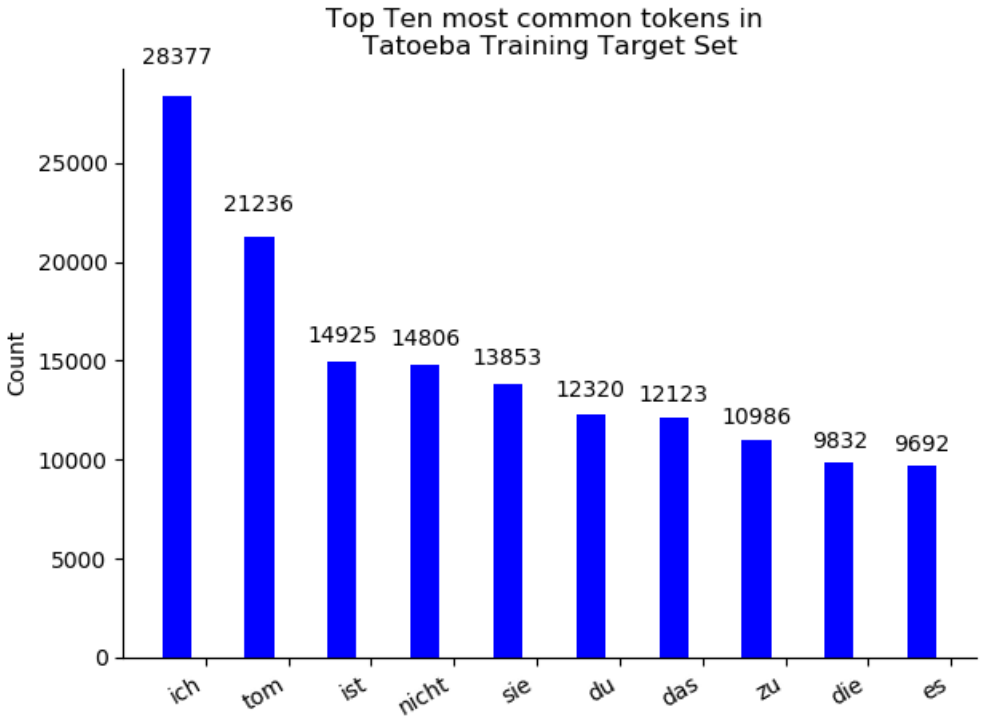


Figure 5.9.: Top Ten tokens in the Tatoeba Training-Taget dataset

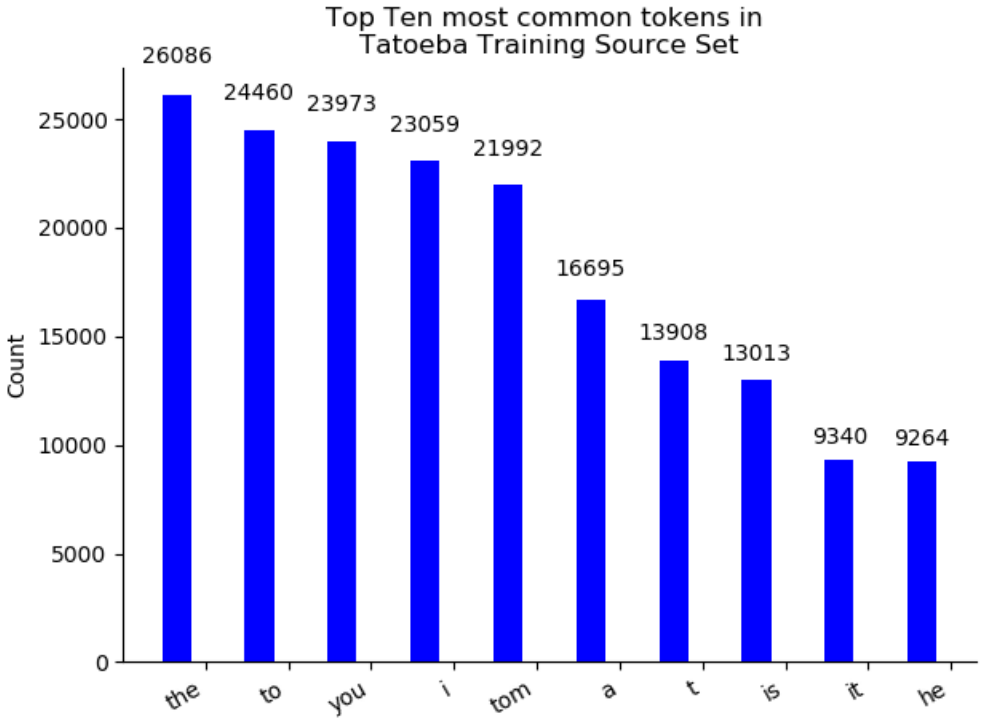


Figure 5.10.: Top Ten tokens in the Tatoeba Training-Source dataset

## 5.2. WMT16

WMT16 is by far the largest dataset we trained our models on. It is a well established dataset in the NMT-Community and consists of a number of multiple corporas and is publicly available at http://www.statmt.org/wmt16/. For our training purposed we only used the English-German datasets. We use newstest2013 as our validation set and newstest2015 as our test set. Table 5.2 gives a good idea of the various split sizes.

| Set | Unique Tokens | Paragraphs | Avg. Length | Max. Length | Not in Vocab. |
|---|---|---|---|---|---|
| Trn. Src. | 781'206 | 4'500'966 | 25.6095 | 139 | - |
| Trn. Tar. | 1'597'025 | 4'500'966 | 24.2873 | 112 | - |
| Val. Src. | 8'780 | 2'998 | 21.6698 | 27 | 1'103 |
| Val. Tar. | 11'982 | 2'998 | 21.2592 | 30 | 3'566 |
| Tes. Src. | 7'359 | 2'170 | 21.7544 | 22 | 1'114 |
| Tes. Tar. | 9'549 | 2'170 | 20.5733 | 32 | 3'112 |

Table 5.3.: Dataset WMT16

It is also interesting to note that the maximum length of paragraphs in the training data is larger by a factor of three to four when compared to the validation and test datasets. However, in order to mitigate the BLEU's bias towards shorter sentences we ensure that our split datasets contained an equal average paragraph length. Lastly, we were surprised that the number of tokens occurring in the validation and test sets but not in the vocabulary remained extremely low, despite the large size of the dataset. Figures 5.11 through 5.14 provide an overview of the tokens with the highest occurrence counts in the validation and test sets of the respective source and target languages, which were not included in the vocabulary. As expected, these terms were often (brand) names, colloquial terminology, geographic names or abbreviations.

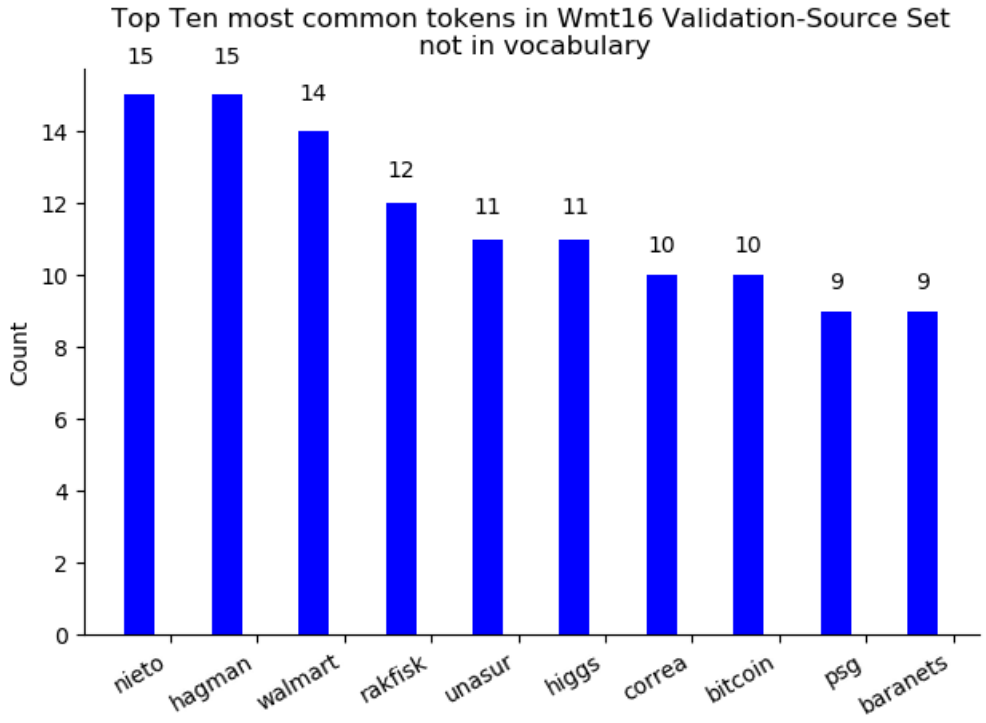


Figure 5.11.: Occurrences of tokens in the WMT16 Validation-Source Set which were not included in the vocabulary

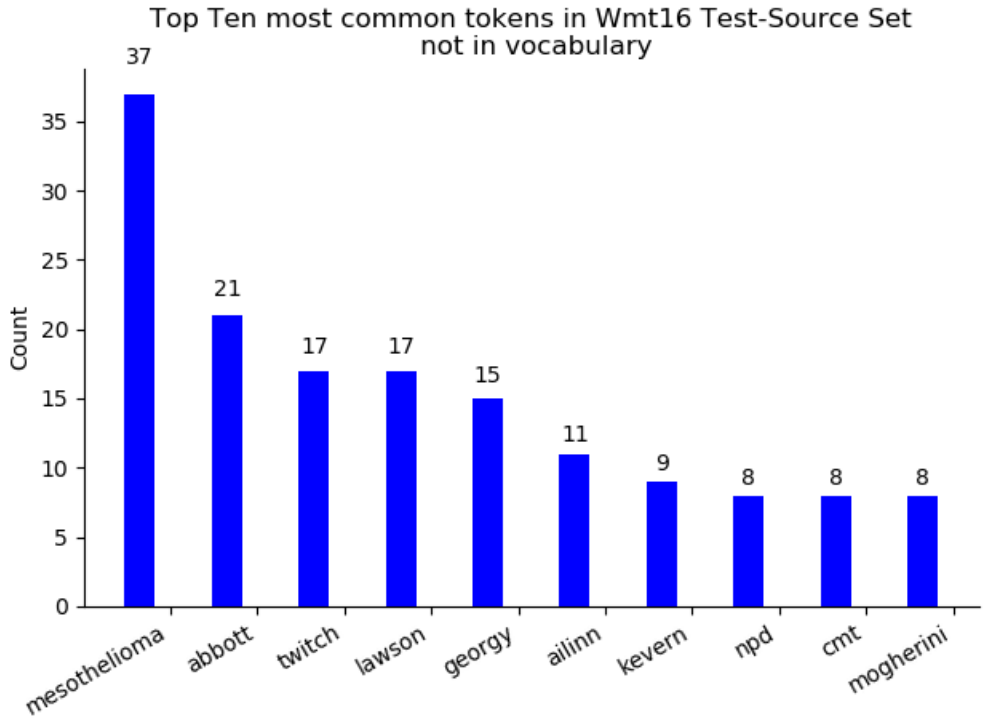


Figure 5.13.: Occurrences of tokens in the WMT16 Test-Source Set which were not included in the vocabulary

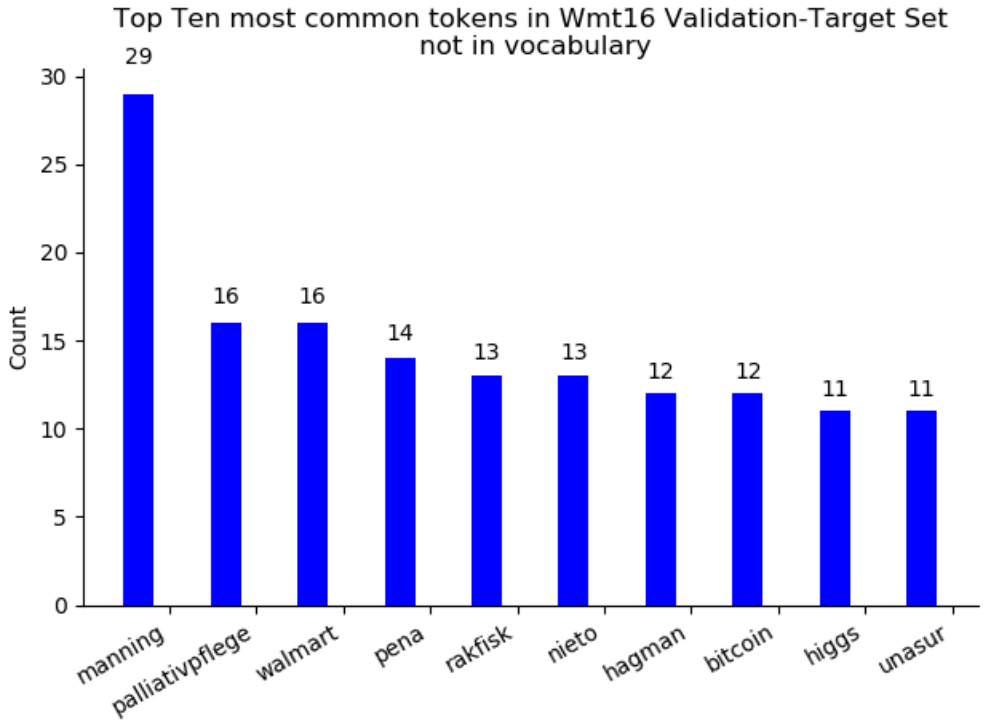


Figure 5.12.: Occurances of tokes in the WMT16 Validation-Target Set which were not included in the vocabulary

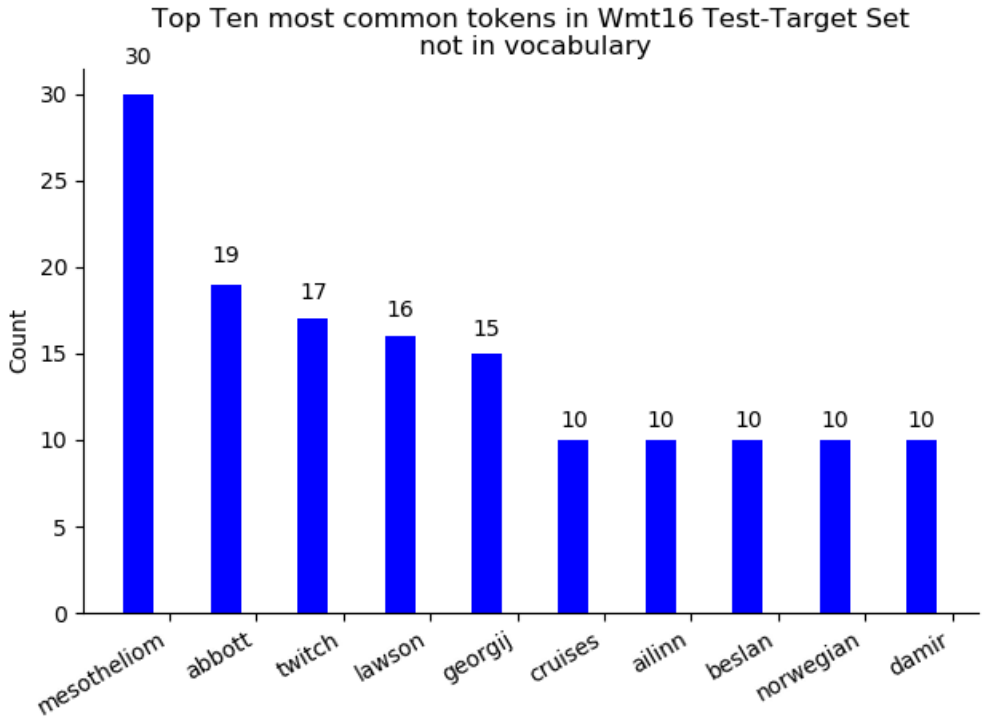


Figure 5.14.: Occurrences of tokens in the WMT16 Test-Target Set which were not included in the vocabulary

Figures 5.15 through 5.20 provide insight into the most common tokens in the respective split target and source data sets. Please note that we have removed punctuations from this list as it would completely dominate every figure.

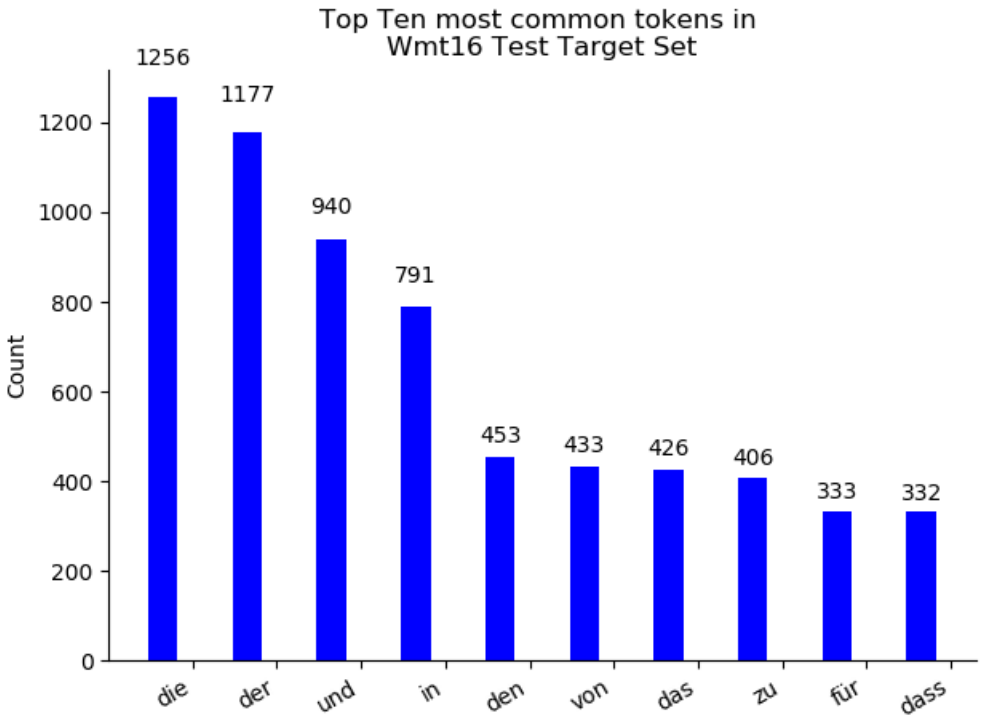


Figure 5.15.: Top Ten tokens in the WMT16 Test-Target dataset

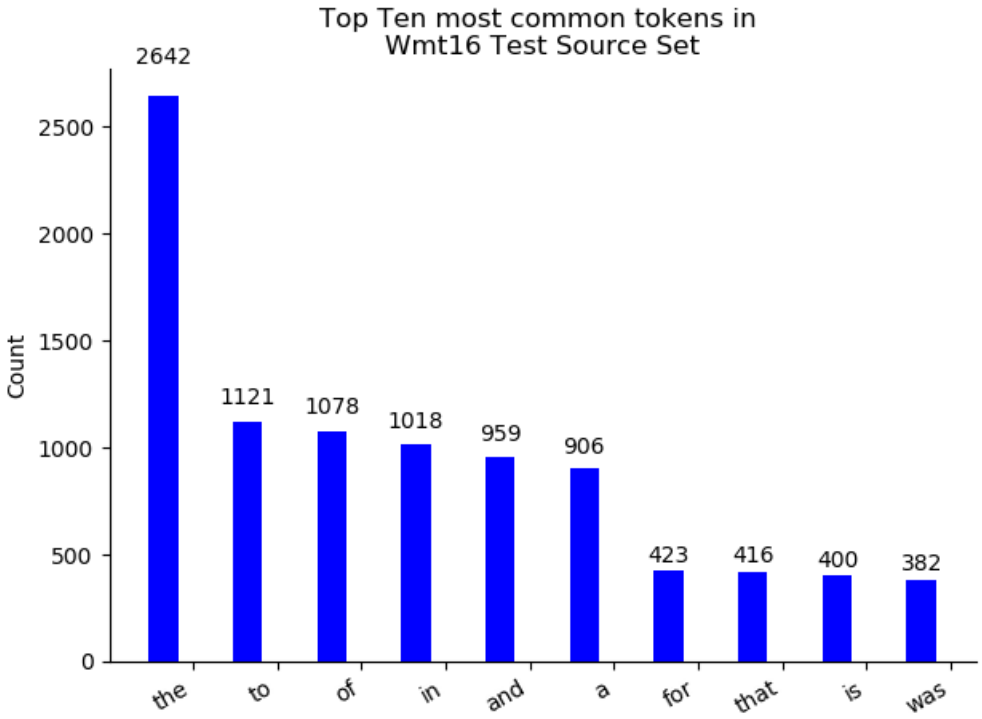


Figure 5.16.: Top Ten tokens in the WMT16 Test-Source dataset

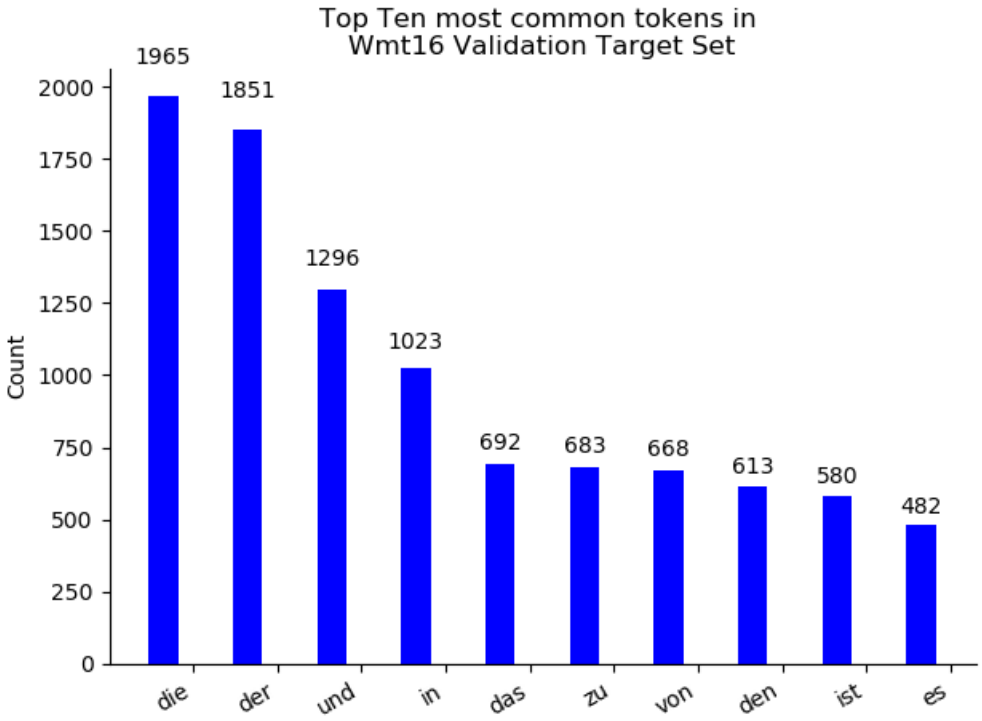


Figure 5.17.: Top Ten tokens in the WMT16 Validation-Target dataset

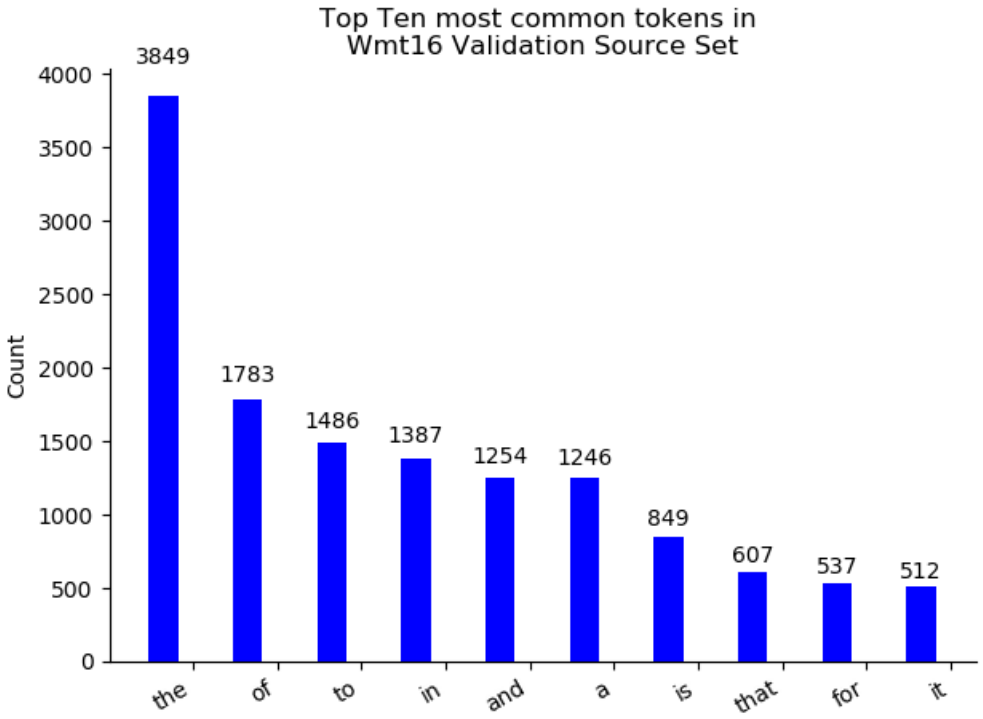


Figure 5.18.: Top Ten tokens in the WMT16 Validation-Source dataset

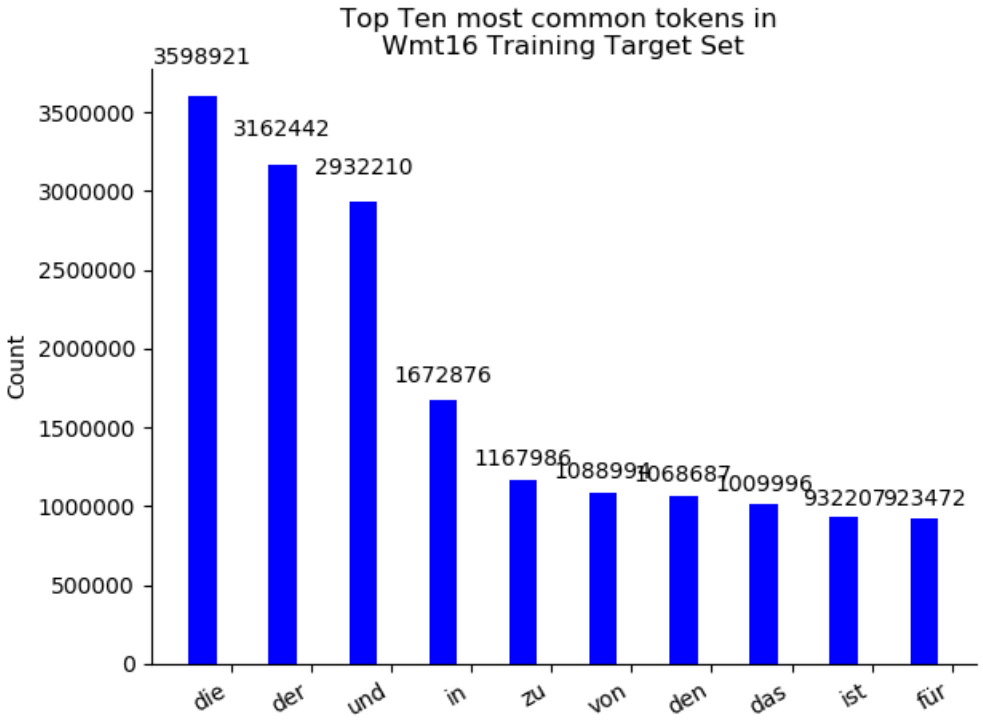


Figure 5.19.: Top Ten tokens in the WMT16 Training-Taget dataset

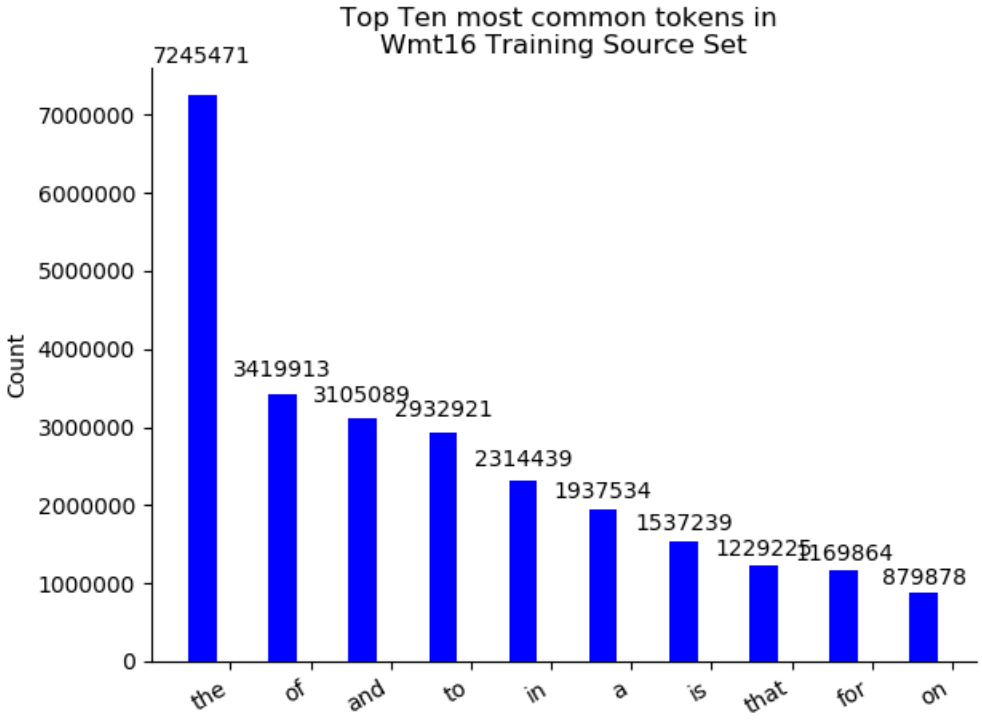


Figure 5.20.: Top Ten tokens in the WMT16 Training-Source dataset

## 5.3. WMT 1 Mio. Data

The WMT 1 Mio. dataset was created in order for us to evaluate performance of the different models with a smaller dataset. It is a subset of the dataset described in 5.2 and as such has similar characteristics. Table 5.3 gives a brief oversight of these.

| Set | Unique Tokens | Paragraphs | Avg. Length | Max. Length | Not in Vocab. |
|---|---|---|---|---|---|
| Trn. Src. | 334'971 | 1'000'000 | 25.6056 | 121 | - |
| Trn. Tar. | 658'758 | 1'000'000 | 24.2788 | 90 | - |
| Val. Src. | 8'780 | 2'998 | 21.6698 | 27 | 1'117 |
| Val. Tar. | 11'982 | 2'998 | 21.2592 | 30 | 3'594 |
| Tes. Src. | 7'359 | 2'170 | 21.7544 | 22 | 1'135 |
| Tes. Tar. | 9'549 | 2'170 | 20.5733 | 32 | 3'130 |

Table 5.4.: Dataset WMT16 1 Mio.

Figures 5.21 through 5.24 provide an overview of the tokens with the highest occurrence counts in the validation and test sets of the respective source and target languages, which were not included in the vocabulary.

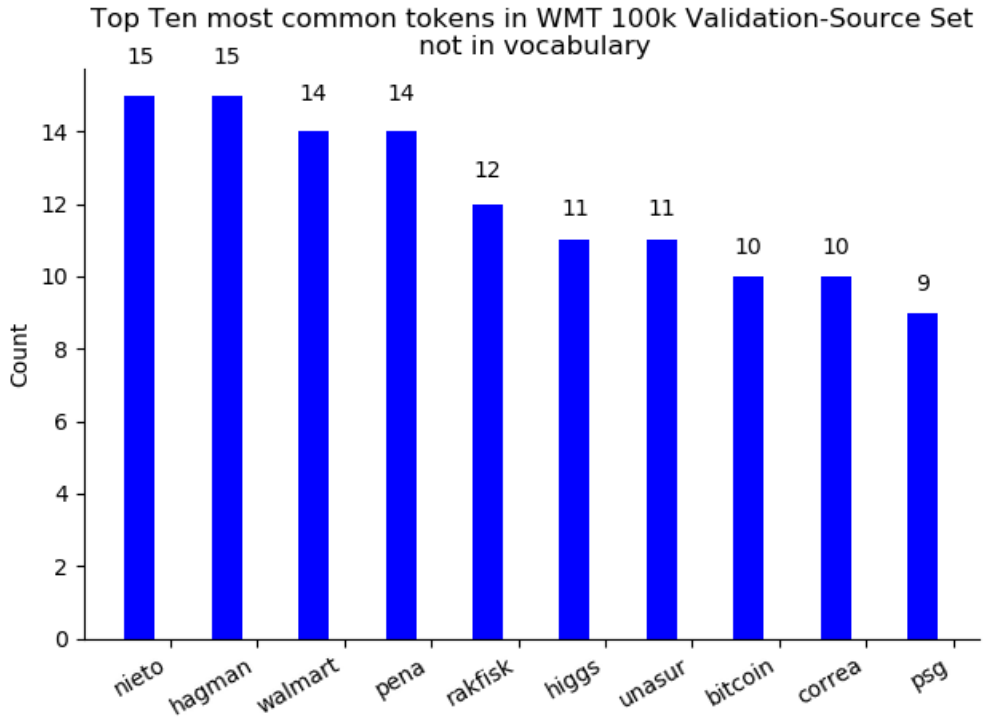


Figure 5.21.: Occurrences of tokens in the WMT16 1 Mio. Validation-Source Set which were not included in the vocabulary

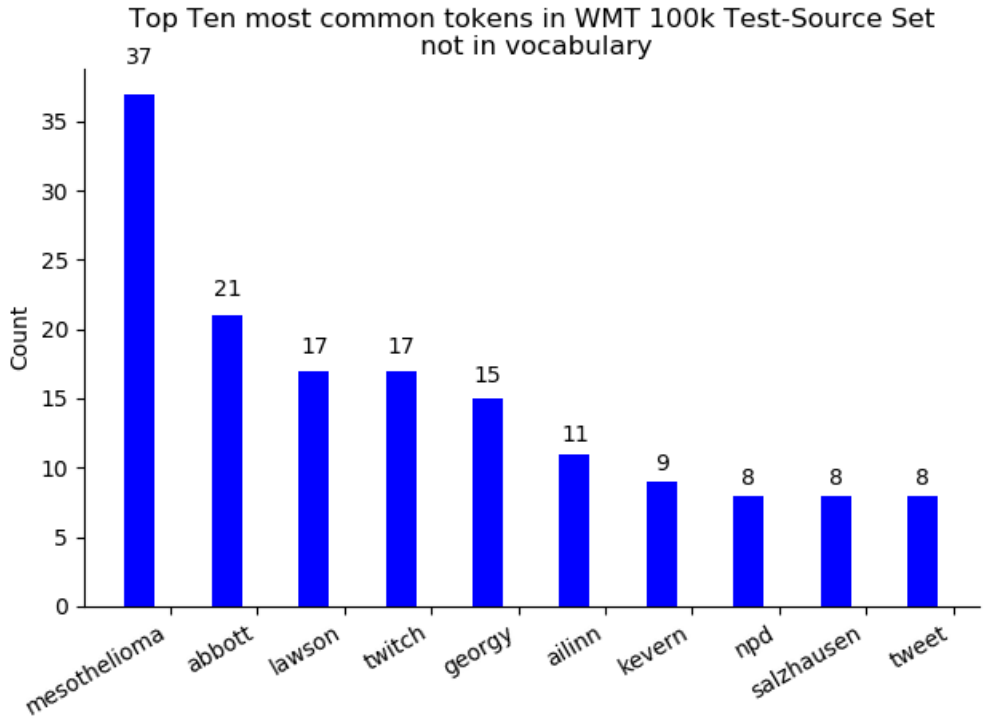


Figure 5.23.: Occurrences of tokens in the WMT16 1 Mio. Test-Source Set which were not included in the vocabulary

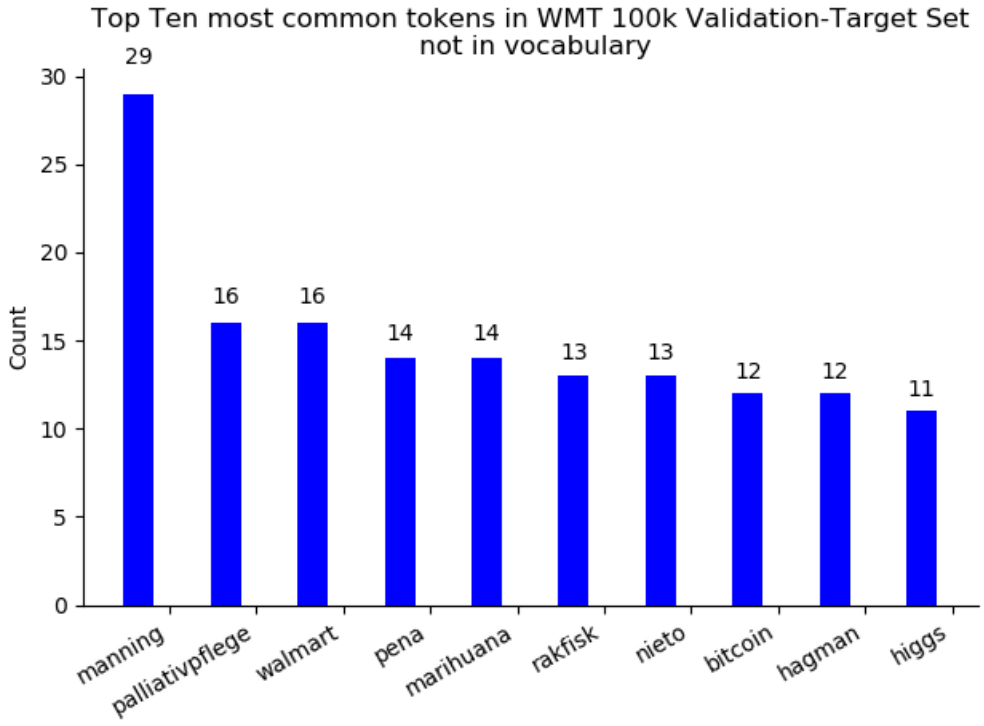


Figure 5.22.: Occurances of tokes in the WMT16 1 Mio. Validation-Target Set which were not included in the vocabulary

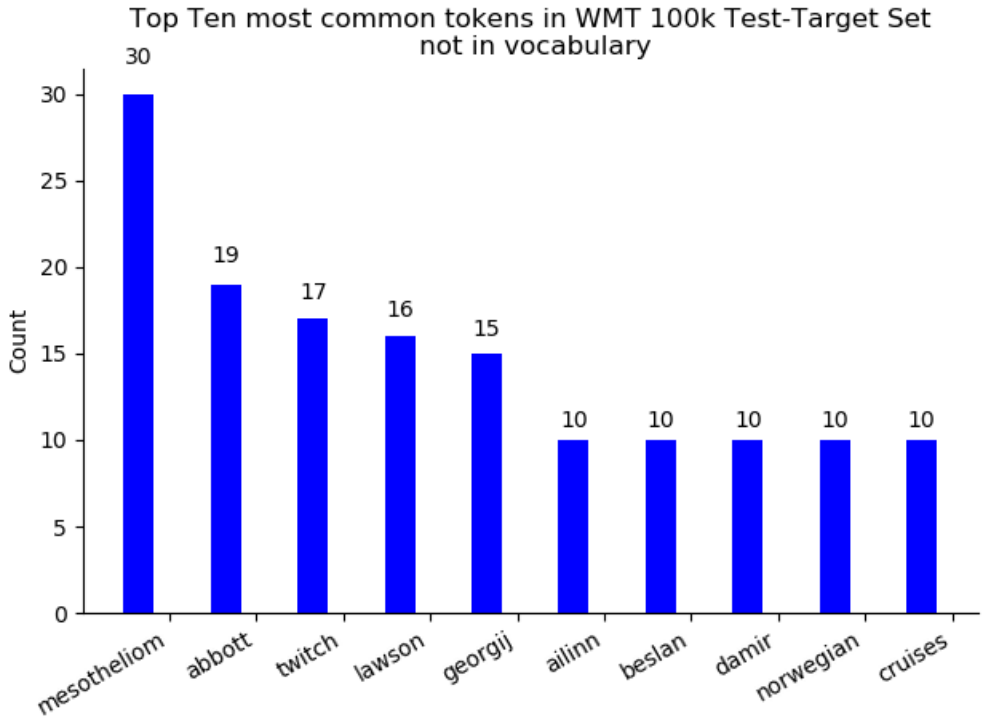


Figure 5.24.: Occurrences of tokens in the WMT16 1 Mio. Test-Target Set which were not included in the vocabulary

Figures 5.25 through 5.30 provide insight into the most common tokens in the respective split target and source data sets. Please note that we have removed punctuations from this list as it would completely dominate every figure.

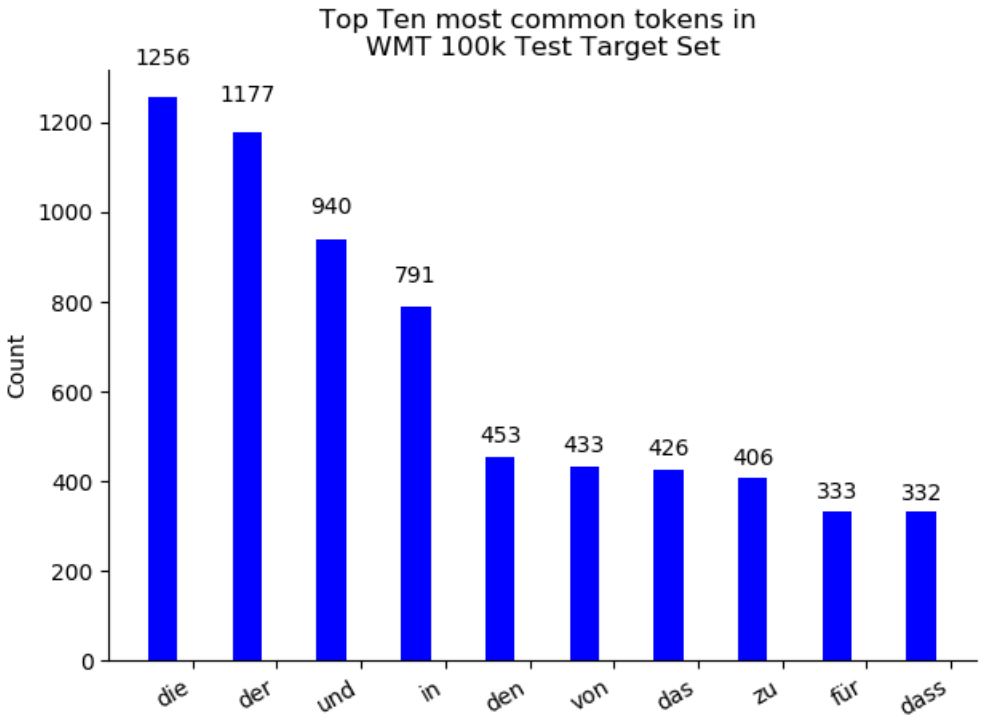


Figure 5.25.: Top Ten tokens in the WMT16 100k Test-Target dataset

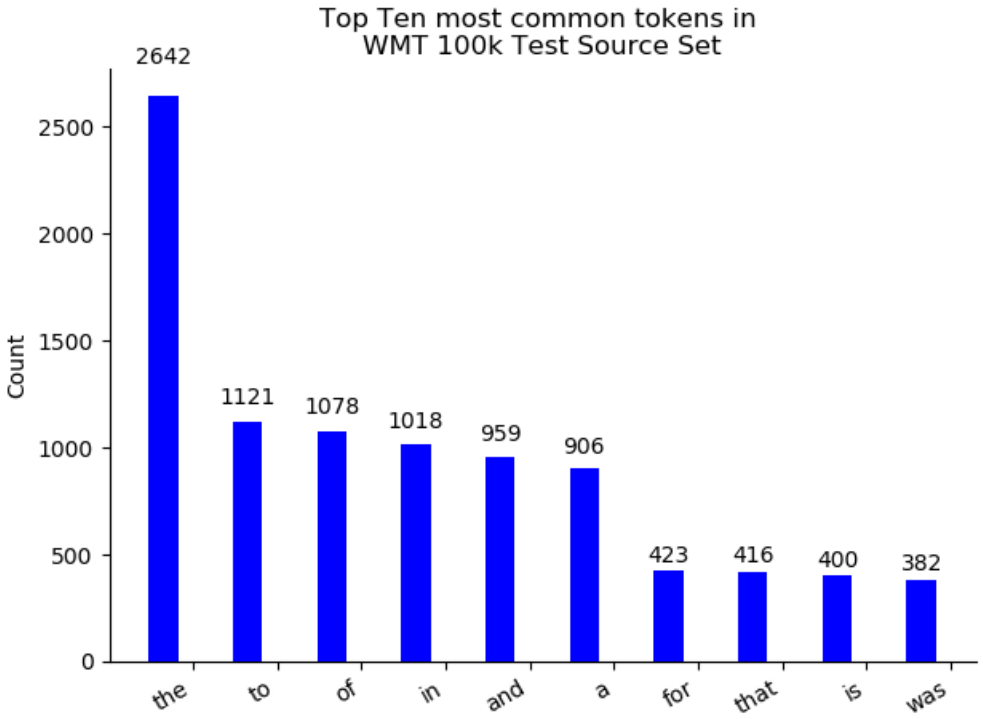


Figure 5.26.: Top Ten tokens in the WMT16 100k Test-Source dataset

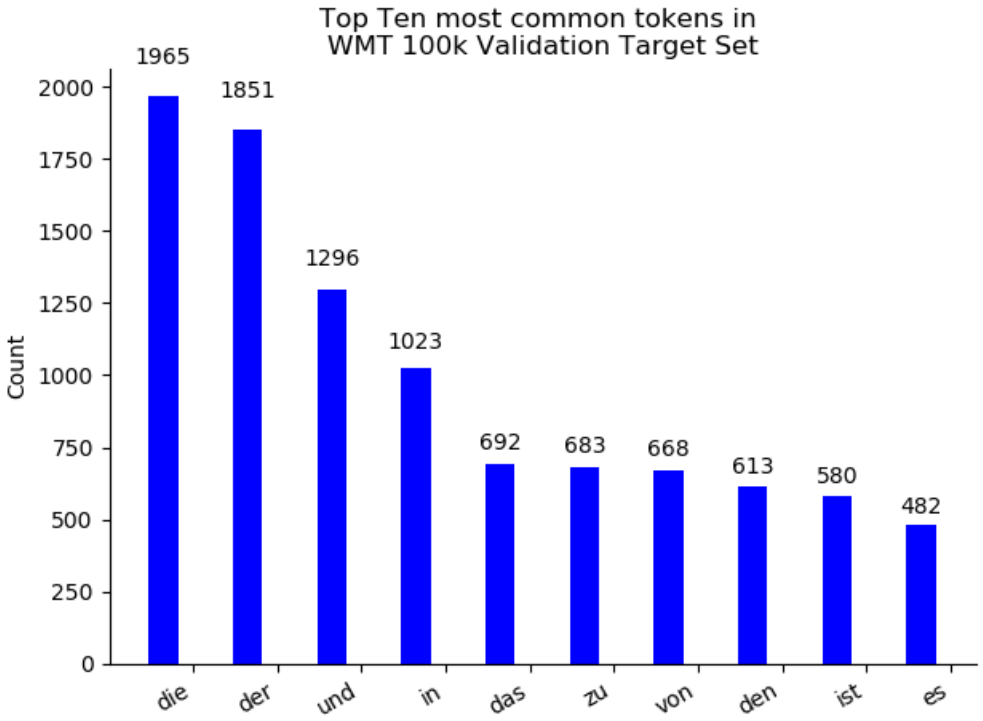


Figure 5.27.: Top Ten tokens in the WMT16 100k Validation-Target dataset

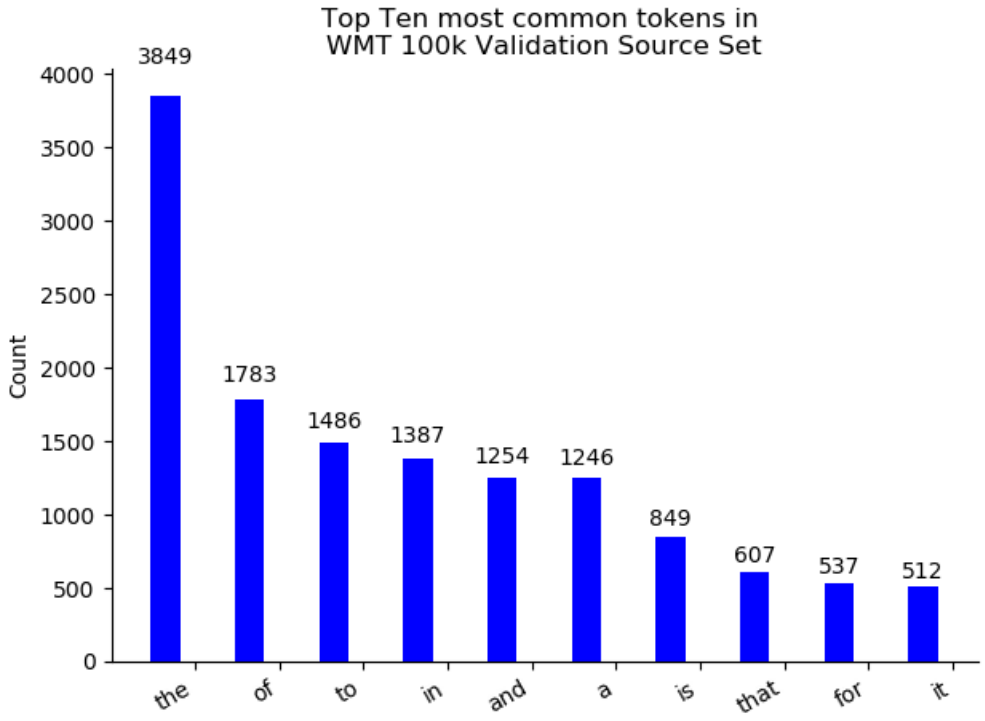


Figure 5.28.: Top Ten tokens in the WMT16 100k Validation-Source dataset

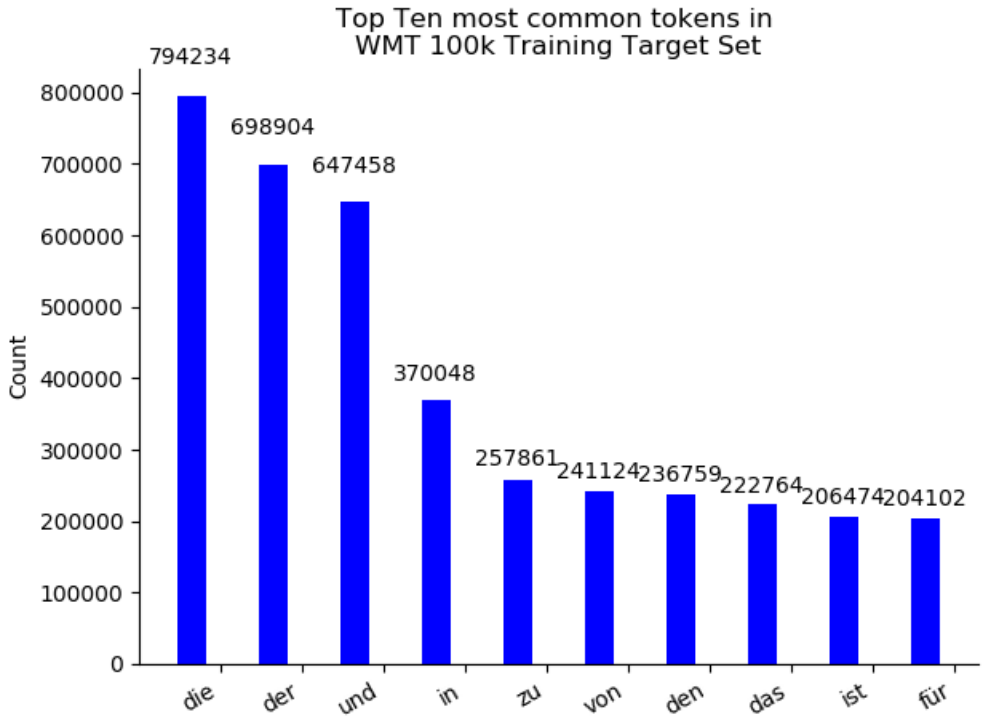


Figure 5.29.: Top Ten tokens in the WMT16 100k Training-Taget dataset

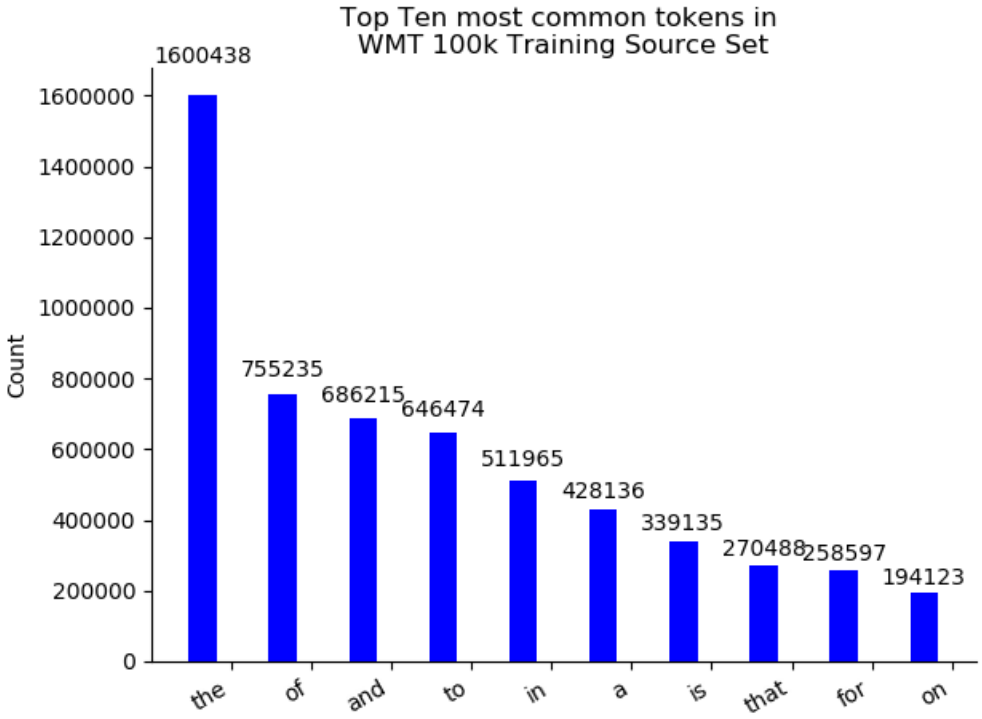


Figure 5.30.: Top Ten tokens in the WMT16 100k Training-Source dataset

## 5.4. FastAI Data

FastAIDataset is an subset of gigafrench release 2, which was released in the WMT09. The subset we used contains only the questions. As this consisted of a English-French corpus we had to split it into a separate training, validation and test set with a respective 70%/20%/10% ratio, as the newstest2013 and newstest2015 datasets are English-German. Table 5.4 provides an oversight of how the dataset was split and the individual characteristics.

| Set | Unique Tokens | Paragraphs | Avg. Length | Max. Length | Not in Vocab. |
|---|---|---|---|---|---|
| Trn. Src. | 13'186 | 18'315 | 7.9735 | 27 | - |
| Trn. Tar. | 26'924 | 18'315 | 7.9147 | 33 | - |
| Val. Src. | 8'780 | 5'234 | 21.6698 | 27 | 925 |
| Val. Tar. | 11'982 | 5'234 | 21.2592 | 30 | 1329 |
| Tes. Src. | 7'359 | 5'233 | 21.7544 | 22 | 954 |
| Tes. Tar. | 9'549 | 5'233 | 20.5733 | 32 | 1385 |

Table 5.5.: Dataset FastAI

Figures 5.31 through 5.34 provide an overview of the tokens with the highest occurrence counts in the validation and test sets of the respective source and target languages, which were not included in the vocabulary. As this dataset is relatively small in comparison to the other, the words appearing in the Figures 5.31 through 5.34 are more common words than (brand) names, locations or colloquial terms, as is usually the case.

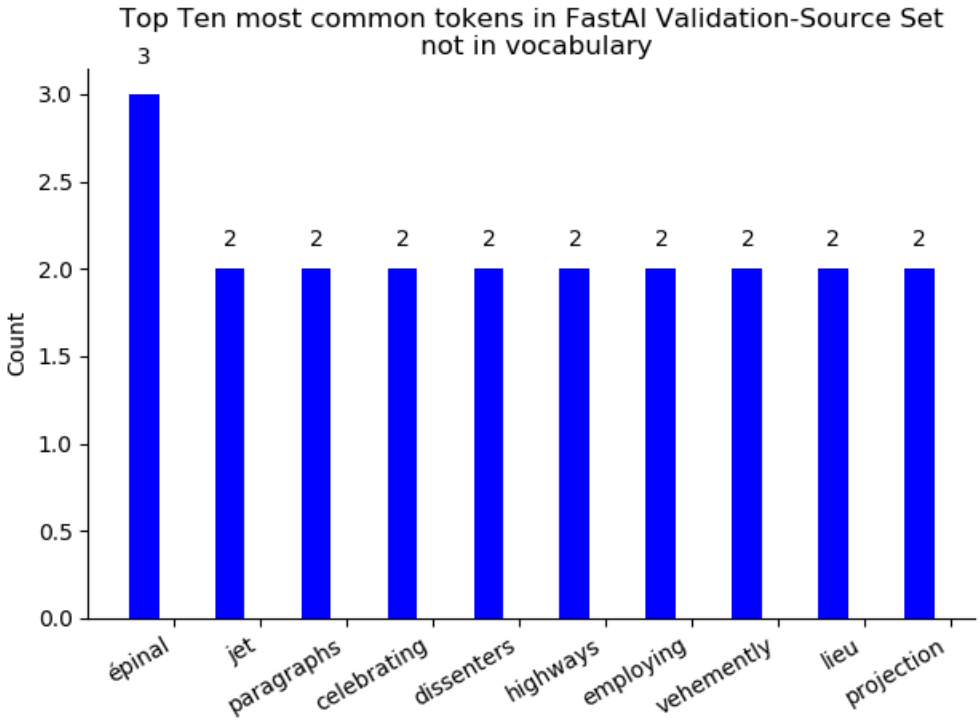


Figure 5.31.: Occurrences of tokens in the FastAI Validation-Source Set which were not included in the vocabulary

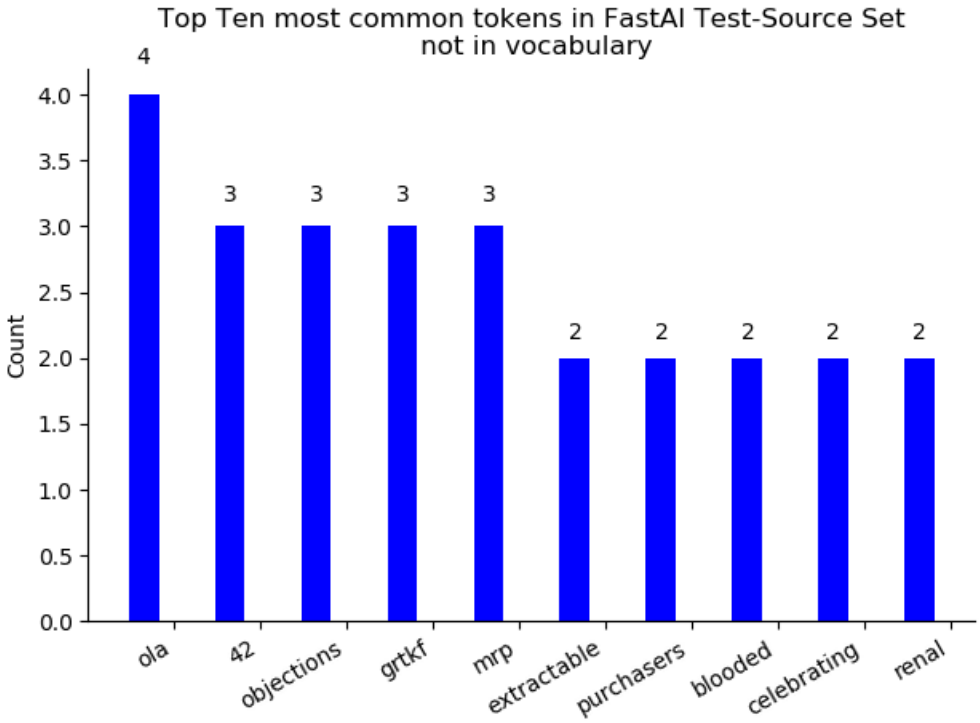


Figure 5.33.: Occurrences of tokens in the FastAI Test-Source Set which were not included in the vocabulary

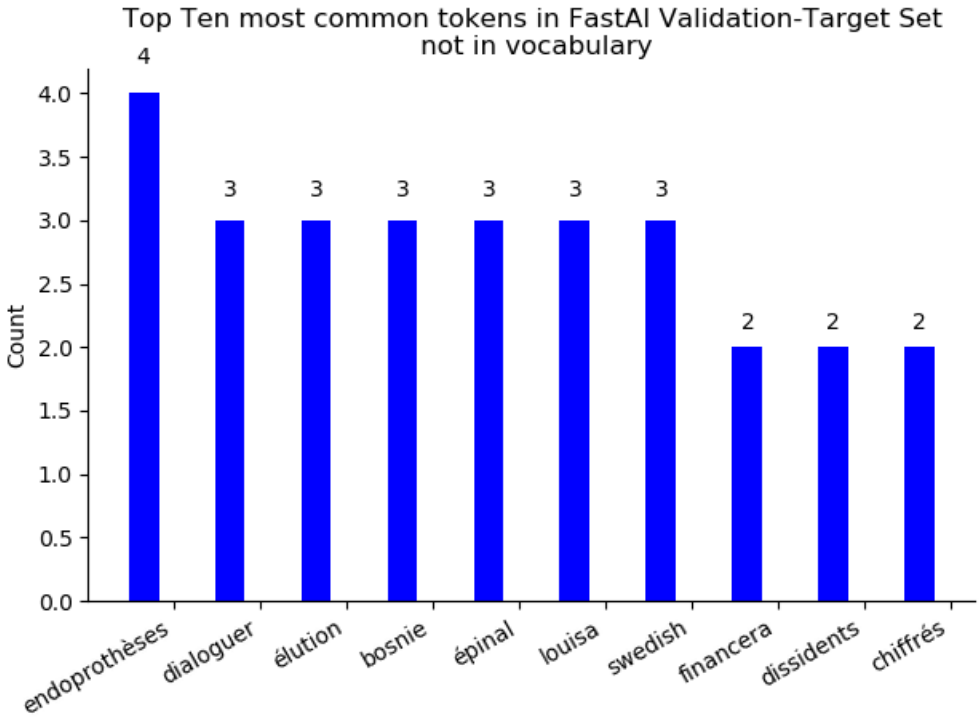


Figure 5.32.: Occurances of tokes in the FastAI Validation-Target Set which were not included in the vocabulary

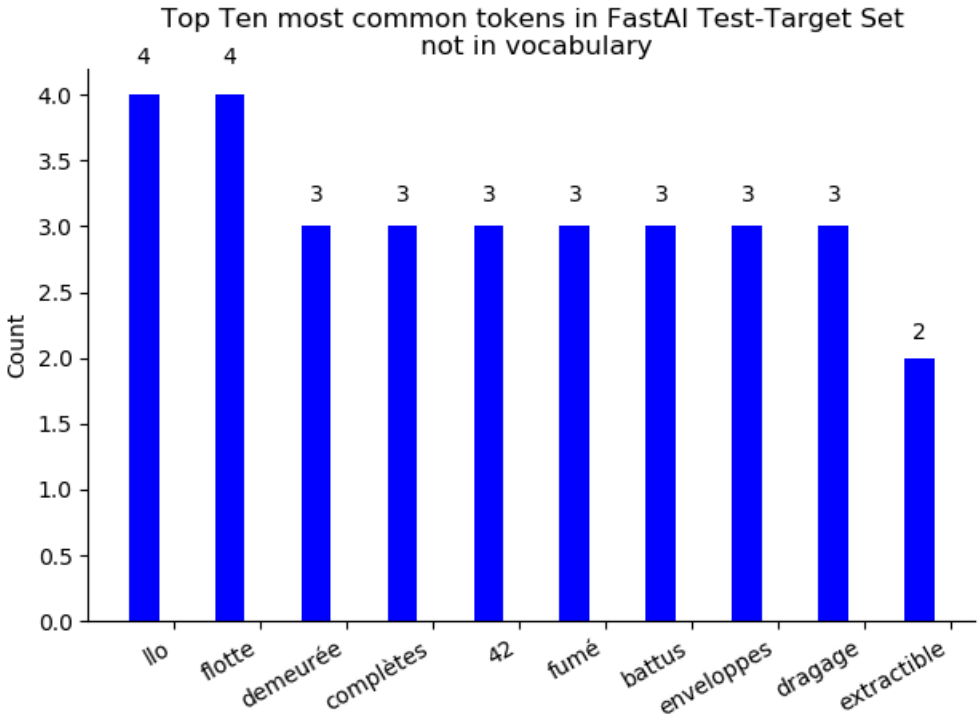


Figure 5.34.: Occurrences of tokens in the FastAI Test-Target Set which were not included in the vocabulary

Figures 5.35 through 5.35 provide insight into the most common tokens in the respective split target and source data sets. Please note that we have removed punctuations from this list as it would completely dominate every figure.

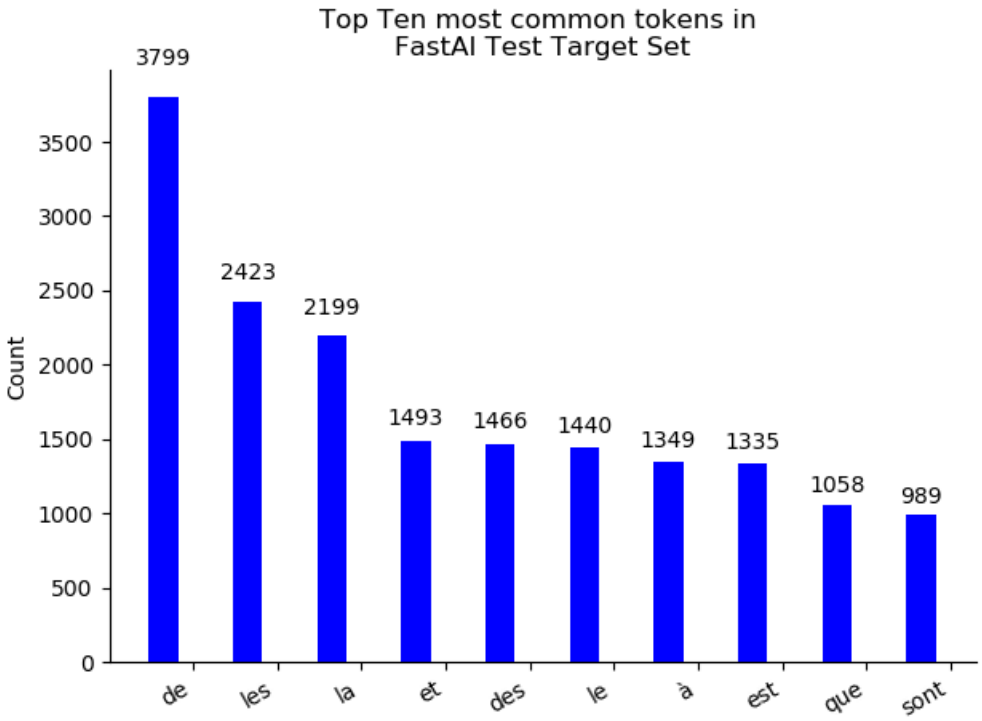


Figure 5.35.: Top Ten tokens in the FastAI Test-Target dataset

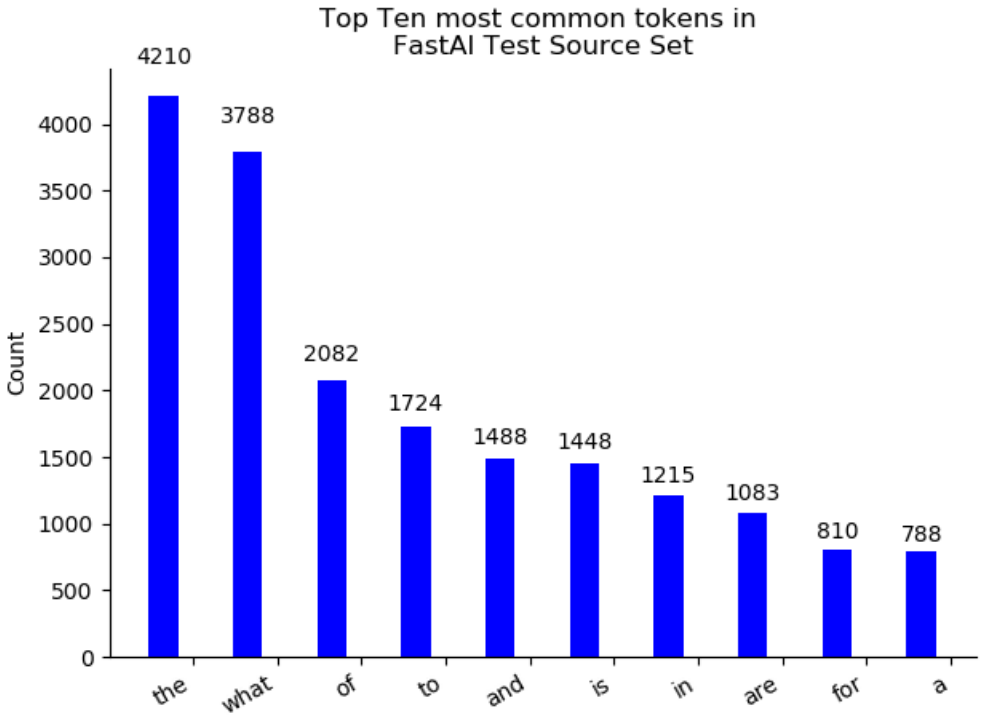


Figure 5.36.: Top Ten tokens in the FastAI Test-Source dataset

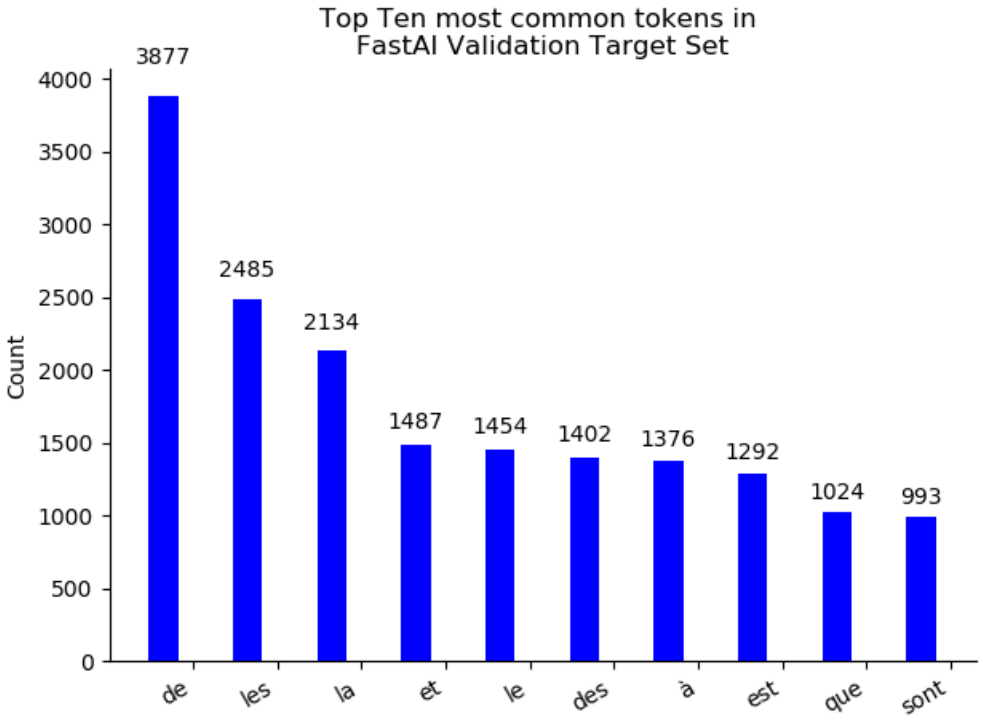


Figure 5.37.: Top Ten tokens in the FastAI Validation-Target dataset

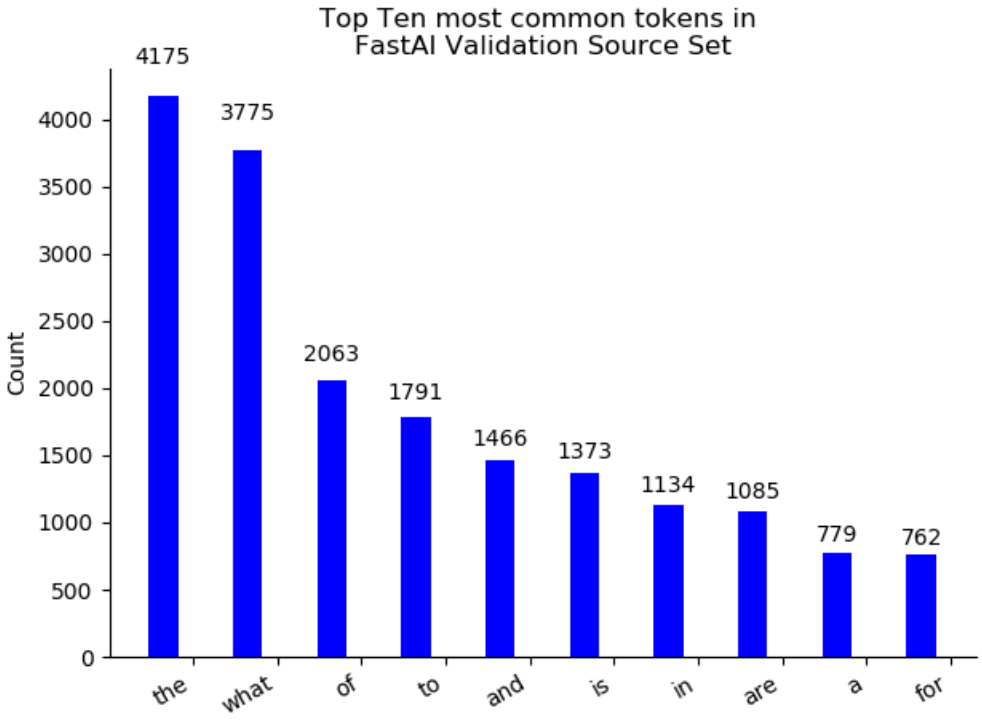


Figure 5.38.: Top Ten tokens in the FastAI Validation-Source dataset

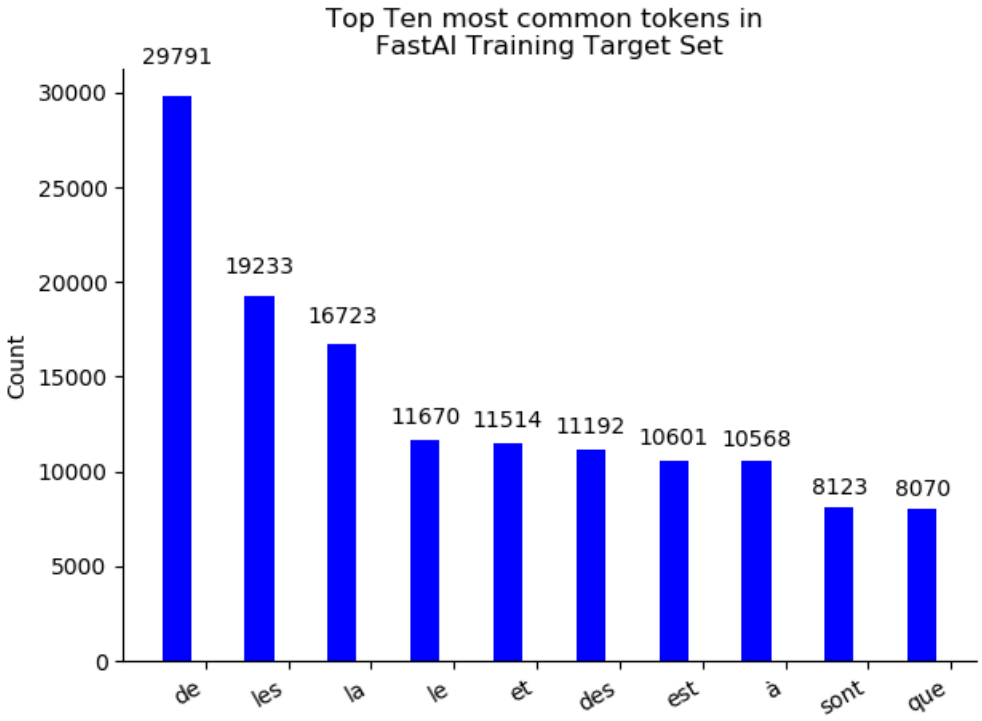


Figure 5.39.: Top Ten tokens in the FastAI Training-Taget dataset

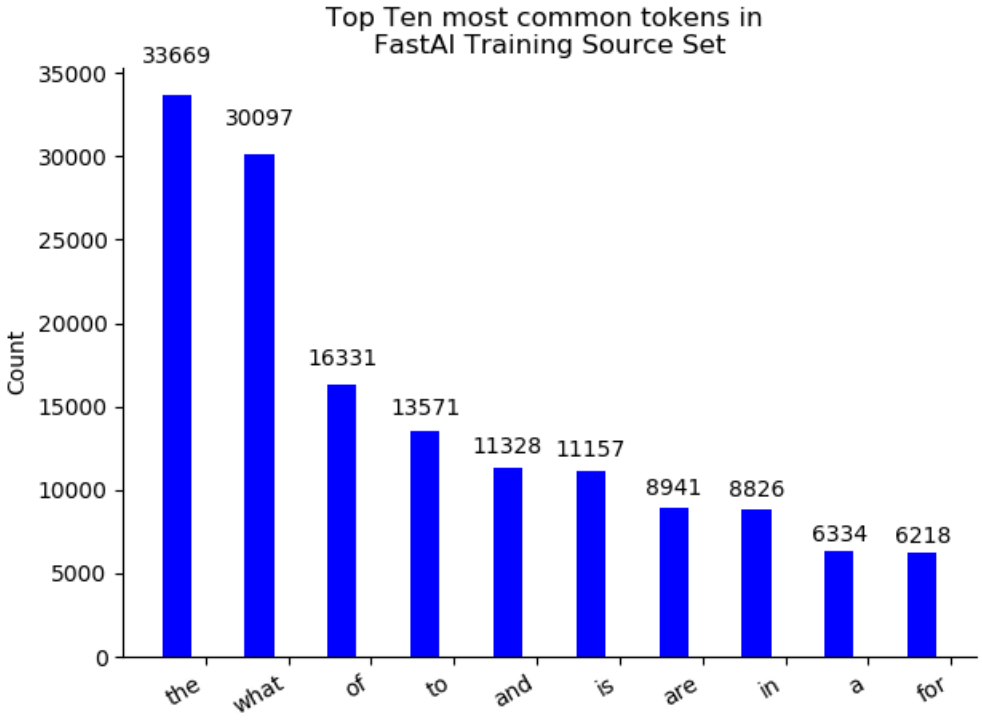


Figure 5.40.: Top Ten tokens in the FastAI Training-Source dataset

## 5.5. Preprocessing

Preprocessing is an important step to get value out of the data. The aim of preprocessing is to transform the data in a suitable format for the given problem. In our case of NMT, it is important to be able to get the right pair of source and target sequence. Machines and also neural networks can't work with text out of the box, so we need to encode text into a vector or number. Existing techniques, such as *word embeddings*, exist to handle this task. The aim of transforming text to *word embeddings* is to represent a word as a high dimensional vector without loosing the accompanied sentiment. As the creation of *word embeddings* is a large tasks in itself, which would have greatly increased the scope of this thesis, as well has the fact that pre-trained *word embeddings* are publicly available, we opted to use pre-trained ones. Exclusively the experiments concerning the Google NMT model, they are trained in parallel to the model.

### 5.5.1. GMNT

Google provides a script for the preprocessing which we used for the training [43]. The script first downloads all the required data, extract it should the datasets be compressed, and concatenates the training data. Once complete, it converts the test and validation set from the sgm format into text files, where each line is a separate translation. All three datasets are tokenized using the Tokenizer from the moses git repository [48]. After the data is tokenized, the model starts to learn the subword units. Once complete, BPE is applied the the vocabulary files are generated. To learn subword units and apply BPE google uses the subword-nmt git repository provided by Sennrich, Rico [49].

### 5.5.2. FastA & ZSOE

The preprocessing for English-German texts for both, the FastAI model and our ZSOE model is identical.First the required data is downloaded, decompressed tokenized and split into the a training-, validation-, and test-dataset with a respective ration of 70%/20%/10%. Following this, the tokens from each set are converted into an id. The id is ascending based on the frequency of each token in the training-set. Proceeding the first token of each translation, a special *start of sequence* token is inserted. The end of each translation is marked by inserting a special *end of sequence* token. Additional we replace tokens which have a higher id than the size of the vocabulary, with another special token, the *unknown* token. In order to solve the issue of varying sequence lengths, padding is applied. We truncate each sequence, which is longer than a defined maximum length. Sequences that are smaller than the defined maximum length are padded with a *padding* token so that their length is equal to the maximum length. The vocabulary is consists of the tokens, which have an id that is lower than the size of the vocabulary. For convenience and performance reasons we opted to save the preprocessed data as a byte stream.

The sgm file format is the raw format of the validation and test set we used. It has a xml structure and containing three different tags. The *srcset*-tag is the outer tag, which holds multiple documents, which, in turn are indicated by a *doc*-tag. Each *doc*-tag groups a text and could contain multiple *seg*-tags. Each *seg* tag is one sentence, which corresponds to a translation in another *.sgm file. The paragraphs are matched across the files using an id-attributes from the *seg*, and *doc*-tags. Listing 5.1 shows the English version of an example *.sgm file and Listing 5.2 shows the corresponding German version.

Listing 5.1: Example english .sgm file

```
<srcset setid="newstest201x" srclang="any">
        <doc docid="basic_example" genre="news" origlang="xx">
                <seg id="1">Hello World</seg>
        </doc>
</srcset>
```

Listing 5.2: Example german .sgm file

```
<srcset setid="newstest201x" srclang="any">
        <doc docid="basic_example" genre="news" origlang="xx">
                <seg id="1">Hallo Welt</seg>
        </doc>
</srcset>
```

The preprocessing of the FastAI Data for the FastAI and ZSOE model is nearly identical to the preprocessing of the English-German data for these models. As shown in section 5, the FastAI Data consists exclusively of the questions from the English-French wmt10 dataset. To accomplish this only the questions of the dataset were filtered out before tokenization.

# 6. Experiments

In this chapter we will describe the methodology of the conducted experiments, as well as present the relevant results.

## 6.1. Compare different implementations

The first type of experiment is aimed at comparing different implementations mentioned in chapter 4. Specifically, the three implementations are as follows:

1. KerasTut - Based off a small tutorial from the *Keras* Blog
2. FastAI - Based off a TensorFlow Talk in Singapur
3. ZSOE - Our own implementation

We did not expect exceedingly high BLEU scores from these models, as they did not make use of many of the previously mentioned advanced techniques, which are common in today's NMT-Models. We have trained and evaluated each model on the four datasets detailed in chapter 5.

**FastAI dataset:** As we can see in table 6.1, the best BLEU score produced by variants of the ZSEO and FastAI Models, which were trained on the FastAI dataset is 1.45. Every FastAI model is better than the corresponding model based on our implementation. This is due to the fact that the FastAI model is more advanced. It uses bi-directional layers and more layers than our model, which enables it to learn more complex functions. However, this increased complexity lets the model overfit. The ZSOE model also overfits, however, the dropout mitigates this to a small degree.

The FastAI model with attention performs worse than the model without attention. This was unexpected and in contrast to the theory, which details that the attention-mechanism should improve the performance of the model [50]. The two models are identical except for the attention mechanism and dropout. The FastAI model with attention was extended with a dropout-mechanism applied after each recurrent layer.

We suspect, that another reason for the worse performance could be, that the model with attention is too complex and requires more training data in order to perform better. We can also see that the FastAI model with 1024 hidden units only achieves a BLEU of 0.26. A direct result of the increased number in hidden units was overfitting, which caused the model to perform poorly on the test data. Our model with 1024 hidden units, only marginally improves the performance from 0.02 to 0.04. The highest BLEU of 1.45 was reached in the 64. epoch. We see in figure 6.1 the BLEU does not increase after this point and starts to fluctuate. Figure 6.3 shows the validation loss and Figure 6.2 shows the training loss over epochs in tensorboard. We can see that the best model ”FastAI” overfits the data, because the validation loss increases after about five epochs. In figure 6.4 we show the learning rate over the epochs, to highlight the decay of the learning rate.

At about epoch 12 the learning rate decreases because the validation loss increases. When the validation loss increases, it could be an indicator that we overshot the minimum, in which we currently are, resulting in a decreased learning rate in order not to overshoot the minimum again.In table 6.1 we also show the duration in seconds per epoch. There we can see that the FastAI model with attention is the slowest and the base FastAI model is the fastest in terms of training time.

| **Model** | **Addition** | **Duration per epoch in sec** | **BLEU** |
|---|---|---|---|
| FastAI | - | 183 | **1.45** |
| FastAI | attention | 334 | 0.66 |
| FastAI | hidden dim 1024 | 218 | 0.26 |
| Our Model | - | 205 | 0.02 |
| Our Model | hidden dim 1024 | 254 | 0.04 |

Table 6.1.: Question dataset

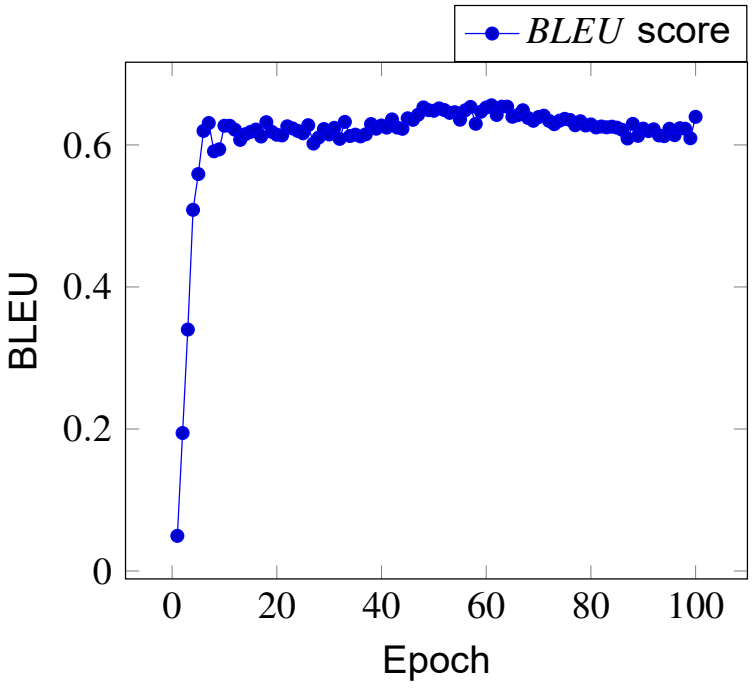


Figure 6.1.: FastAI model Bleu over epochs on the question dataset.

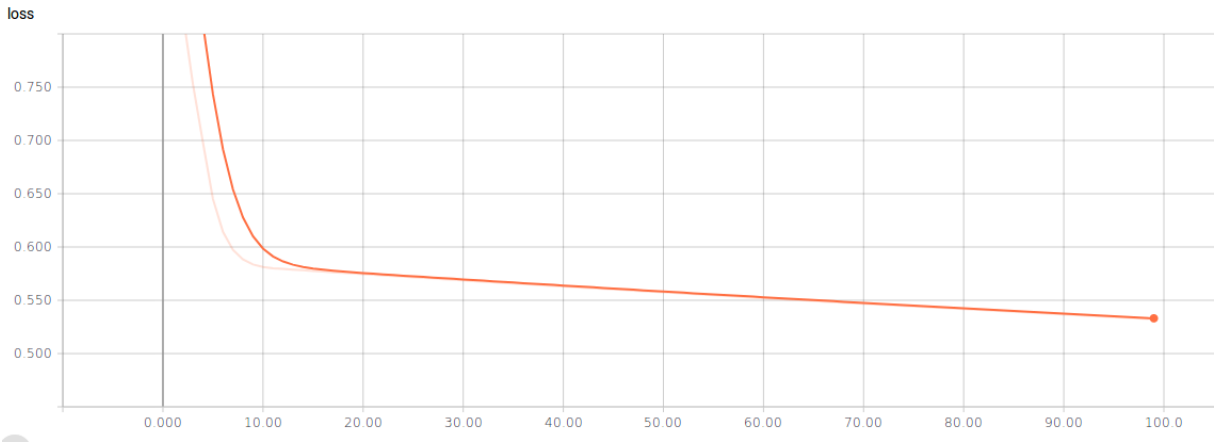


Figure 6.2.: Training loss over epochs of the FastAI model on the question dataset. (Taken from tensorboard, x-axis=epoch, y-axis=loss)

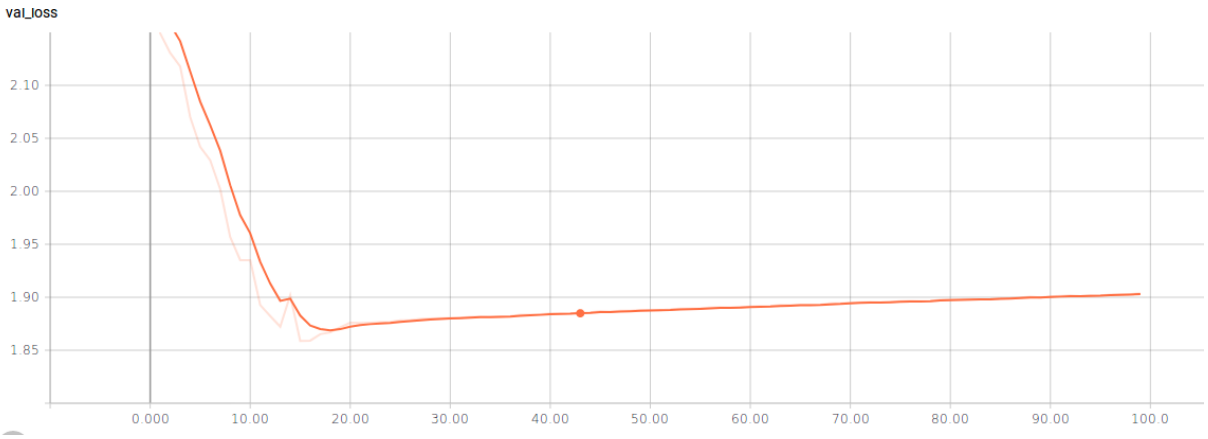


Figure 6.3.: Validation loss over epochs of the FastAI model on the question dataset. (Taken from tensorboard, x-axis=epoch, y-axis=loss)

**Tatoeba dataset:** While training the models on the tatoeba dataset, we can see that our models are better than FastAI due to the fact that we use dropout and our model does not overfit as much as the FastAI model. As on the FastAI dataset, the FastAI Model with attention is not as good as without attention, but our model with more hidden dimensions performs now slightly better than with 128 units. The model with more hidden dimension based on FastAI is still worse than the model with 128 hidden units. The best BLEU score is reached with our model with 1024 hidden dimensions, but the BLEU is still not good enough to produce satisfactory translations.

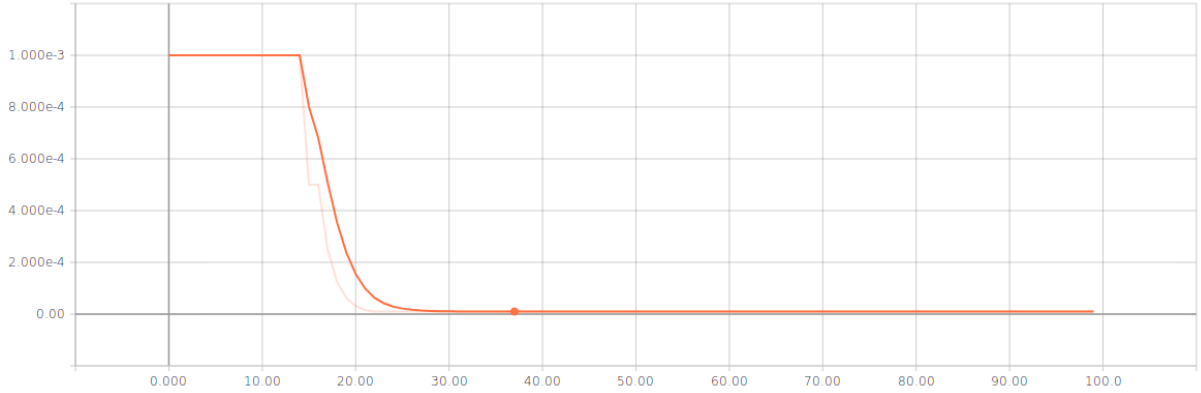


Figure 6.4.: Learning rate over epochs of the FastAI model on question dataset. (Taken from tensorboard, x-axis=epoch, y-axis=loss)

Overall the best BLEU on this dataset is worse than on the question dataset. This comes from the fact, that questions are simpler to translate, because many questions starts and contains the same words like ”What/How/Where” for the English language.

We can also see the slowdown when we use attention in the model or when more hidden units are used. Here our model with only 128 hidden units is the fastest model during training, but the FastAI model is almost as fast as our model.

| **Model** | **Addition** | **Duration per epoch in sec** | **BLEU** |
|---|---|---|---|
| FastAI | - | 457 | 0.66 |
| FastAI | attention | 952 | 0.05 |
| FastAI | hidden dim 1024 | 615 | 0.50 |
| Our Model | - | 449 | 0.69 |
| Our Model | hidden dim 1024 | 730 | **0.74** |

Table 6.2.: Tatoeba dataset

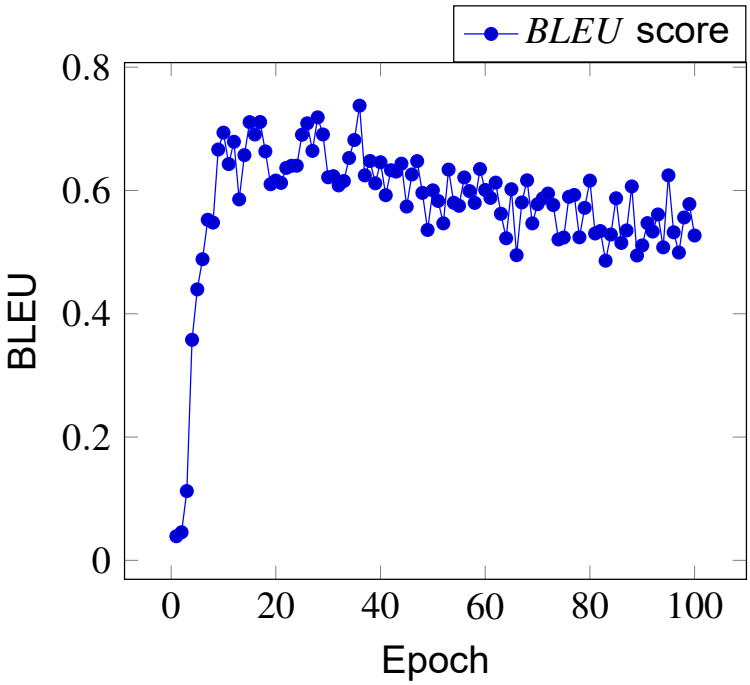


Figure 6.5.: Our model 1024 hidden units Bleu over epochs on the tatoeba dataset.

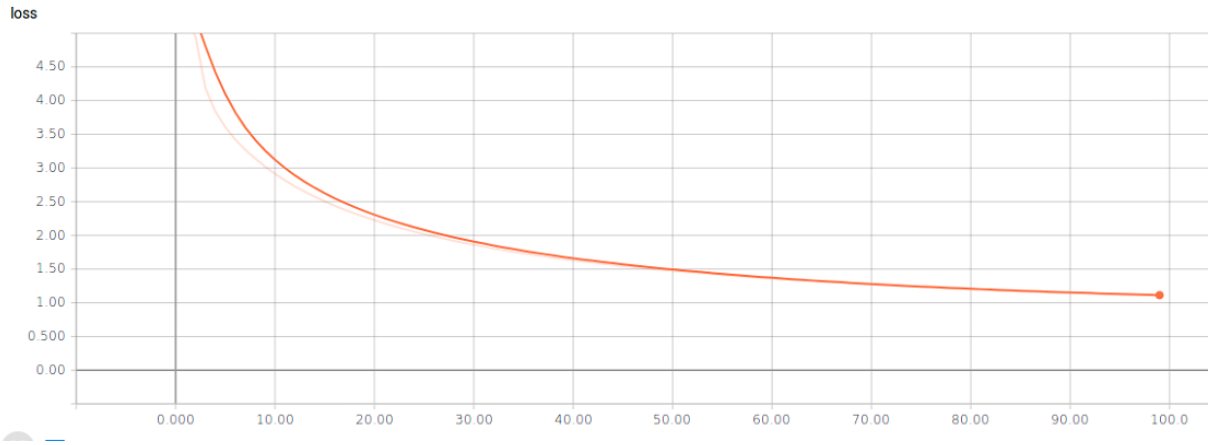


Figure 6.6.: Training loss over epochs of our model on the tatoeba dataset. (Taken from tensorboard, x-axis=epoch, y-axis=loss)

**WMT16 1 Mio.:** When we train the models on 1 million sentences of the WMT16 dataset the BLEU score of every model increases, that proves that more data helps in general to build a good NMT system. The main reason for the increase in BLEU is the increase in trainings data, which leads to a bigger vocabulary size, as shown in section 5 this dataset has 334'971 unique tokens in the source language

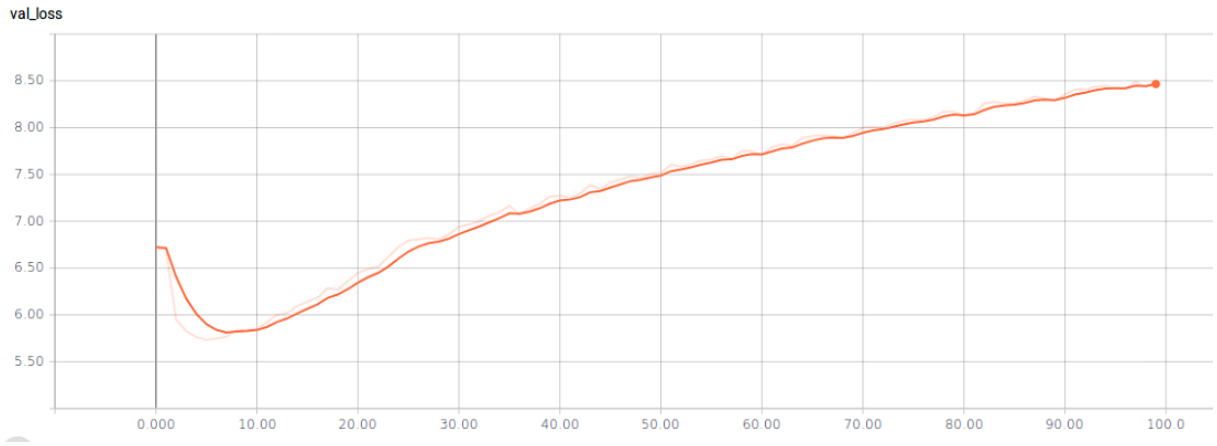


Figure 6.7.: Validation loss over epochs of the our model with 1024 hidden units on the tatoeba. (Taken from tensorboard, x-axis=epoch, y-axis=loss)

and 658'758 unique tokens in the target language, to speed up training and due to limited memory we reduce the vocabulary size to 40'000 words, for each language we use a separate vocabulary. The larger training data and vocabulary leads to out-of-vocabulary words, so that the NMT system is not longer able to understand each token of the input sequence and cannot produce the correct translation.

In terms of training time, our model and the FastAI model with 128 hidden units are also the fastest and the FastAI model with attention is the slowest. The models with 1024 hidden units are between the fastest and slowest model.

| **Model** | **Addition** | **Duration per epoch in sec** | **BLEU** |
|---|---|---|---|
| FastAI | - | 4918 | 1.81 |
| FastAI | attention | 7725 | 1.24 |
| FastAI | hidden dim 1024 | 6295 | **2.23** |
| Our Model | - | 4846 | 1.20 |
| Our Model | hidden dim 1024 | 6583 | 1.71 |

Table 6.3.: WMT16 1 mio

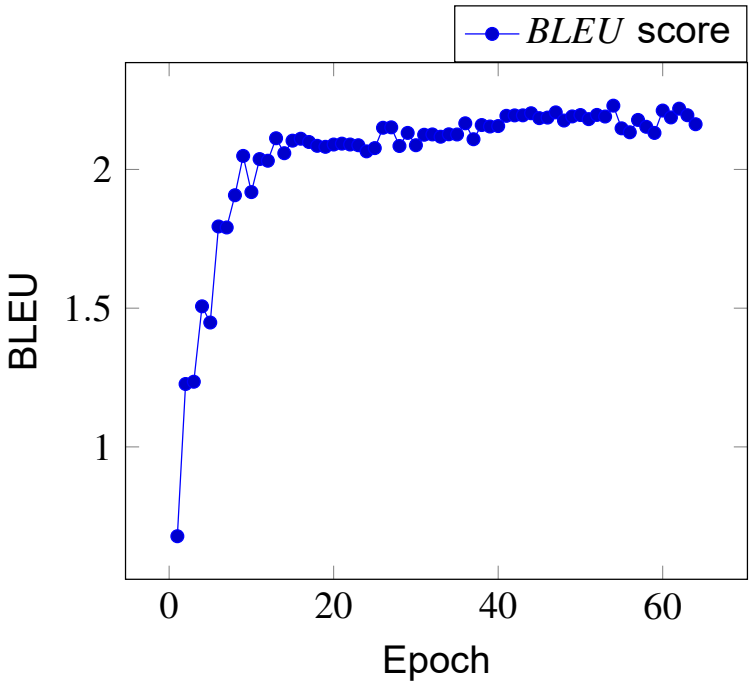


Figure 6.8.: FastAI model 1024 hidden units Bleu over epochs on the WMT16 1 mio. dataset.

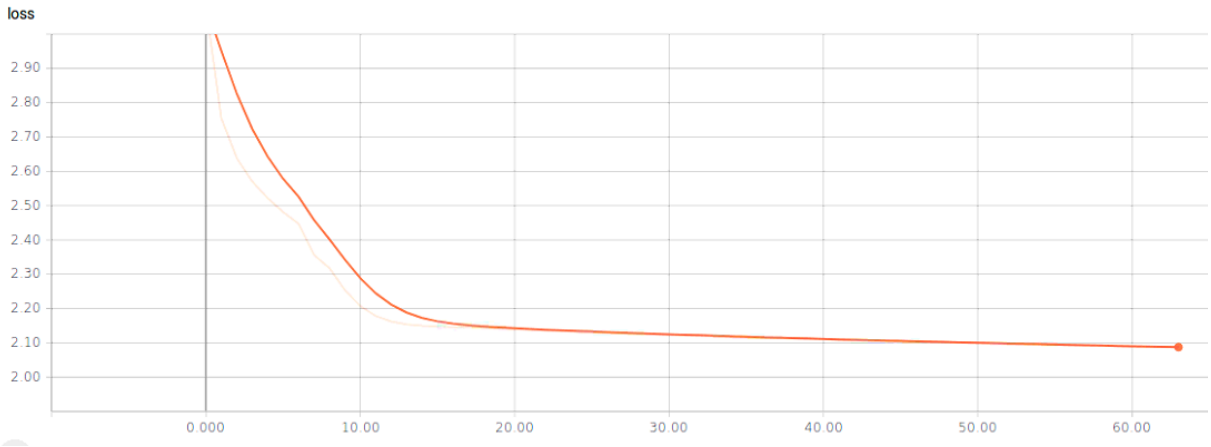


Figure 6.9.: Training loss over epochs of the FastAI model with 1024 hidden units on the WMT16 1 mio. dataset. (Taken from tensorboard, x-axis=epoch, y-axis=loss)

**WMT16 big:** The last dataset on which we train our models is the WMT16 (big). Here the performance measured in BLEU of every model improves compared to the scores on the WMT16 1mio. dataset. In table 6.4 we can see that the performance of the FastAI model without attention is still better as with attention, which is strange, as attention should improve the performance, we think that the model

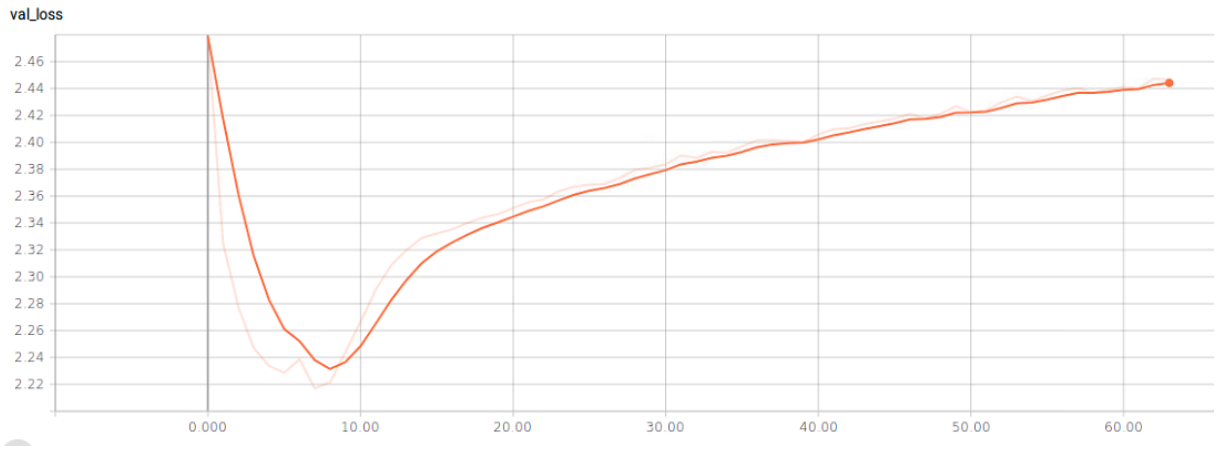


Figure 6.10.: Validation loss over epochs of the our model with 1024 hidden units on the WMT16 1 mio. dataset. (Taken from tensorboard, x-axis=epoch, y-axis=loss)

with attention has some implementation errors, because it is the same model as without attention and it uses dropout, but both changes should improve the performance.

The models with more hidden units have a batter BLEU score, which shows, that the models capture more information during encoding and so the decoded sequences are based on a better thought vector, which results in the performance increase.

We can also see that the bidirectional layers in the FastAI model improving the performance compared to unidirectional layers, like we use them in our model.

We can now also see that the FastAI model with attention is the slowest model and our model with 128 hidden units is the fastest one. Compared to the WMT16 1mio dataset, the models training time is about five times higher, where the dataset size is about 4.5 times higher. Some models are even slower than five times, like the FastAI model with 1024 hidden units, which is about 6.6 times slower than on the WMT16 1 Mio. dataset.

Figure 6.5 shows some example translations with the best model from these experiments, the FastAI model with a hidden dimension of 1024 and a BLEU of 3.25. We can see that the model does not understand each word, but the model captures that it have to produce question marks when the source sentence is a question. We also see that the model performs bad on long sentences.

| **Model** | **Addition** | **Duration per epoch in sec** | **BLEU** |
|---|---|---|---|
| FastAI | - | 24960 | 2.51 |
| FastAI | attention | 44215 | 2.32 |
| FastAI | hidden dim 1024 | 41873 | **3.25** |
| Our Model | - | 22223 | 1.35 |
| Our Model | hidden dim 1024 | 41673 | 1.97 |

Table 6.4.: WMT16 big

| Source sentence | Target sentence | Translation of NMT system |
|---|---|---|
| a republican strategy to counter the re-election of obama | eine republikanische strategie , um der wiederwahl von obama entgegen-zutreten | eine , strategie obama die der wahl |
| I enjoy it. | Ich genieße es. | ich habe es genieße. |
| Take the test or not? | Sollte der Test gemacht werden oder nicht? | nehmen sie das testen? |
| How are you ? | Wie geht es dir? | Wie geht es ihnen? |
| the same applied to hispanics . | gleiches gilt für die hispanische bevölkerung . | gleiches gleiche . . . |
| students, voters considered to be voting more for democratic candidates, are not allowed in several states to use the photo id card issued by their institution. | den studenten, die als wähler angese-hen werden , die ihre stimme vermehrt für demokratische kandidaten abgeben, ist es in mehreren staaten nicht erlaubt , ihren von ihrer einrichtung ausgestellten lichtbildausweis zu benutzen. | studenten , wähler wähler wähler wähler wähler wähler kandidaten stimmen stim-men , in in in , , , , , visitenkarten , ihre ihre ihre . . |

Table 6.5.: Example translation with the FastAI hidden dim 1024 model.

## 6.2. Understand the thought vector

To understand the meaning of the thought vector we encode the hidden state of each sequence in the validation set of the best model on the WMT16 dataset. Additional, we choose some sample sentences, which are shown in table 6.6 and encode them into the hidden state. Then we reduce the dimensionality of each thought vector and plot the additional sentences in a three dimensional plot to if similar sentences are grouped together in space. Figure 6.11 shows the plot of the sample sentences from table 6.6 and figure 6.12 shows a plot where we randomly select about 30 sentences from the validation set, which we also plot. We can see that the FastAI model with a BLEU of 3.25 is not able to group the encoded sentences of the source language based on their meaning. But we can see that it groups the sentences based on the sentence length, longer sentences are higher at the plot.

Table 6.6.: Sample sentences for which we plot their hiddenstate after dimensionality reduction with PCA.

| **Sentence** |
|---|
| I was given a card by her in the garden. |
| In the garden she gave me a card. |
| She gave me a card in the garden. |
| In the garden I gave her a card. |
| I gave her a card in the garden. |
| She was given a card by me in the garden. |
| Mary admires John. |
| Mary is in love with John. |
| John admires Mary. |
| Mary respects John. |
| John is in love with Mary. |
| John respects Mary. |
| Hello Tom. |
| How are you? |
| Hi. |
| Go away. |
| Go away! |
| how are you? |
| hi Tom. |
| Hello. |
| Hi. |
| In the house. |
| In the building. |
| Which of these methods would be better or more Pythonic. |
| There appears to be two different ways to convert a string . |
| What is the weather like? |

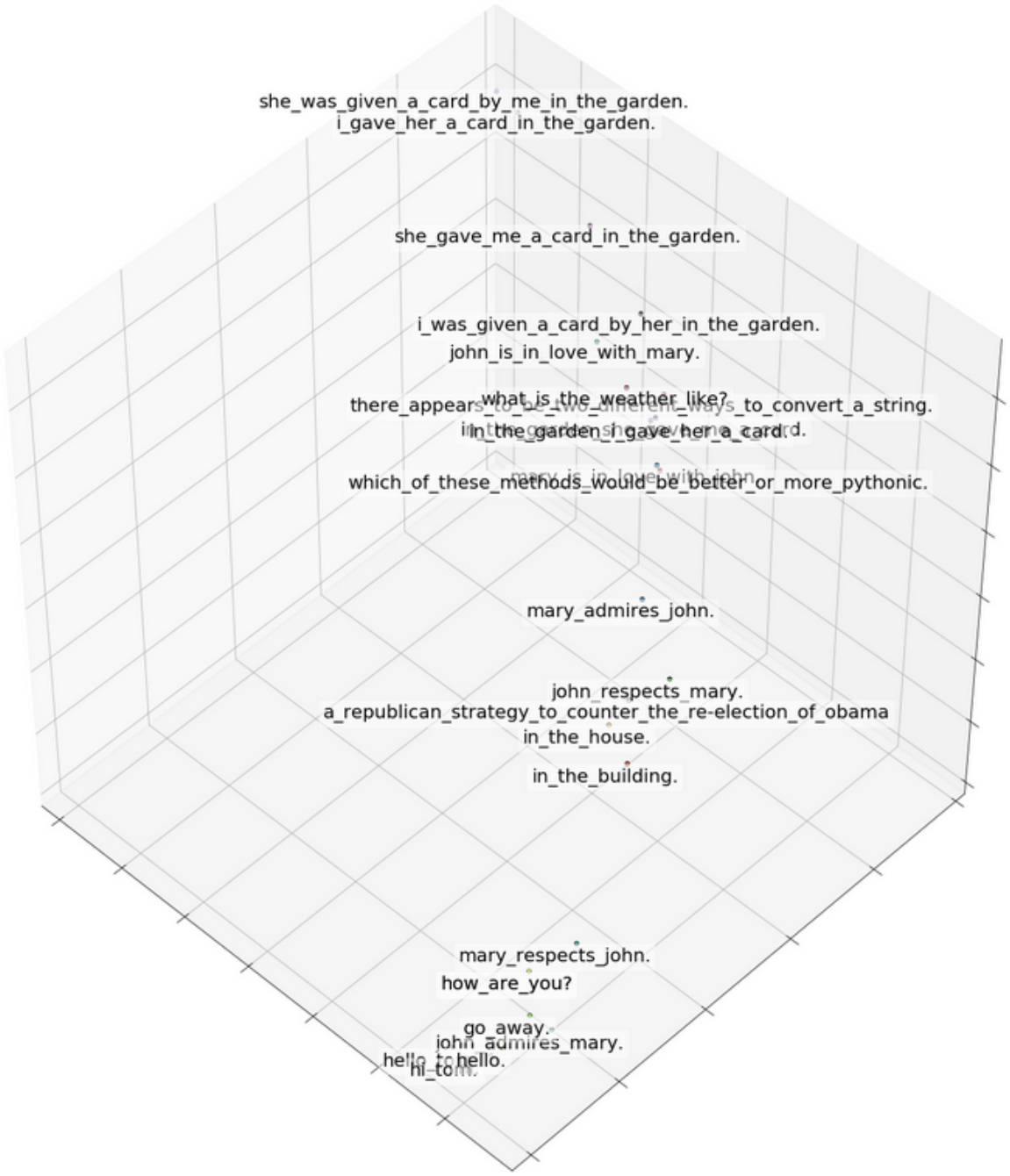


Figure 6.11.: Plot of encoded sentences (thought vector) of the FastAI model with 1024 hidden units on the WMT16 dataset. 26 sentences.

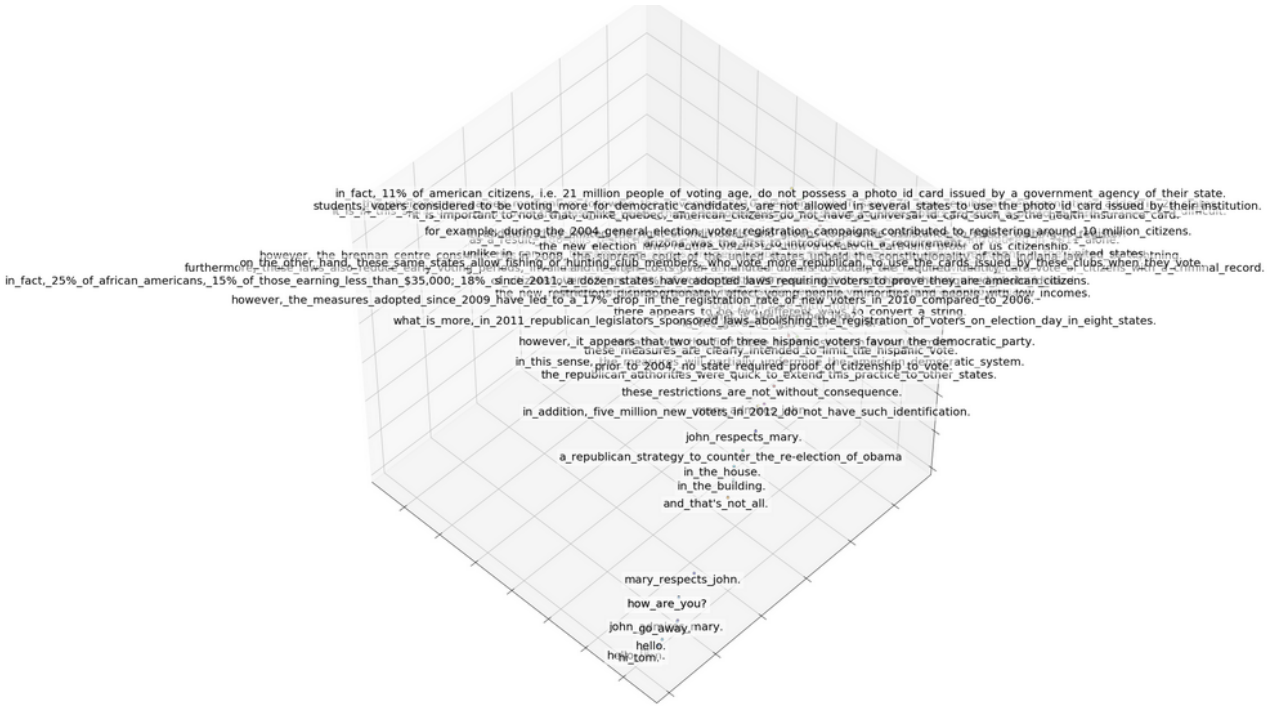


Figure 6.12.: Plot of encoded sentences (thought vector) of the FastAI model with 1024 hidden units on the WMT16 dataset. 60 sentences.

## 6.3. Parameter Exploration

In the following experiments we compare different types of hyper-parameters and their influence on the performance of a NMT system.

### 6.3.1. Baseline Model

Our baseline model is based on the ”en”-”de” model of the NMT Tf-seq2seq framework. We have chosen this because our research has shown that the model implements a lot of state-of-the-art techniques and tricks and it also provides the ability to use different types for the hyper-paramters. With our goal to find out how effective the different settings are, this model provides a great opportunity to concentrate us on the impact of the hyper-parameters rather than implement them ourselves. Because translation is an open-vocabulary problem, this model encodes rare and out-of-vocab words as sequences of sub words. We learn shared sub word units using Byte Pair Encoding with 32’000 merge operations. Our final vocabulary size is 36’548. The hyper-parameter *number of layers* is the amount of recurrent layers in the whole system has. When eight layers are specified, the encoder and decoder will both have four layers. If the number of layers is odd the encoder gets one layer more then the decoder. In table 6.7 we show the hyper-parameters of our baseline model.

Table 6.7.: Google NMT model hyperparameters

| **Parameter** | **Value** |
|---|---|
| Number of units (encoder & decoder) | 1024 |
| Dropout rate | 0.2 |
| Beam width | 10 |
| Gradient norm max value | 5 |
| Initial learning rate | 1 |
| Max sequence length | 50 |
| Number of epochs | 10 |
| Number of layers | 10 |

Figure 6.13 shows the BLEU of our baseline model over steps, where approximately 35'000 steps are equivalent one epoch.

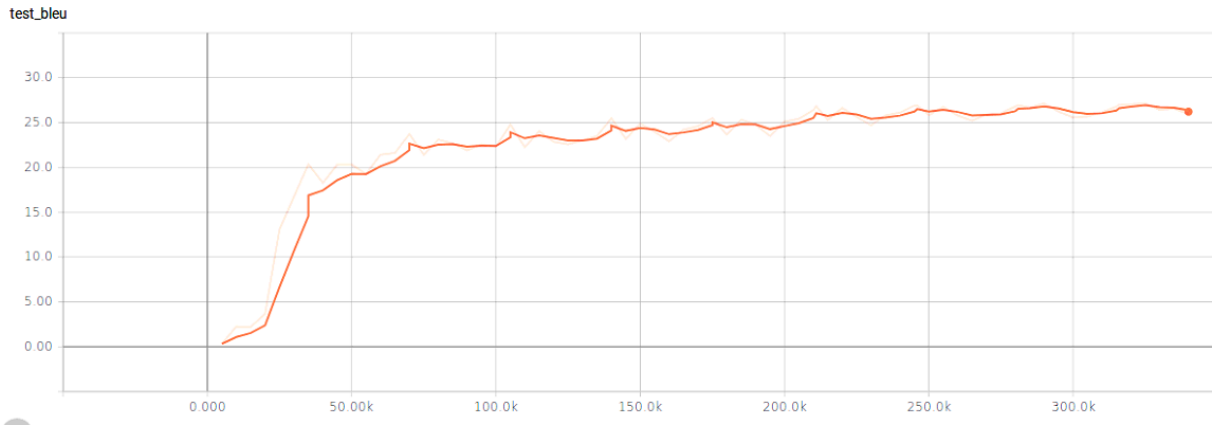


Figure 6.13.: BLEU over steps of our baseline model. (Taken from tensorboard, x-axis=training steps, y-axis=BLEU)

Table 6.8 shows which hyperparameters and their values which we want to change in the following experiments, but because we have not enough time and computational ressources, we need to limit our experiments even more, so we only trained the model with the marked values.

Table 6.8.: Experiments

| **Parameter** | **Variables** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Dataset Size | 0.1 *m* | 0.5 *m* | 1 *m* | 2 *m* | 3 *m* | 4.5 *m* | | | |
| Reverse Input | True | False | | | | | | | |
| Attention Mechanism | **Luong** | **Scaled Luong** | **Bahdanau** | **Normed Bahdanau** | None | | | | |
| Attention Architecture | | **GNMT** | **GNMT v2** | | | | | | |
| Batch Size | 1 | 8 | 16 | 32 | 64 | 128 | | | |
| Number of Units | **128** | **256** | **384** | **512** | **640** | **768** | **869** | **1024** | **1152** |
| Epochs | **1** | **5** | **10** | 15 | 20 | 25 | 30 | 35 | |
| Tokenization | Char-Based | Word-Based | | | | | | | |
| Beam Width | **0** | **4** | **8** | **12** | **16** | 20 | 24 | 28 | |
| Subword | None | BPE | | | | | | | |
| Number of Layers | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | |
| Encoder Types | GNMT | Uni | Bi | | | | | | |
| Vocabulary Size | 20 *k* | 30 *k* | 40 *k* | 50 *k* | 60 *k* | 70 *k* | 80 *k* | 90 *k* | |

### 6.3.2. Attention Mechanism

Here we explore the effect of different attention mechanism on the performance of NMT systems. Since it is not quite clear how important the attention mechanism is we test different mechanisms and found that the Bahdanau attention mechanism is slightly better than the Luong one. We also found that the scaled or normed version of the attention mechanism improved the performance as well. In table 6.9 we present the BLEU score of the different attention mechanisms. The best BLEU is achieved in the baseline model. The improvement of the BLEU over steps for the baseline model can be seen in Figure 6.13.

Table 6.9.: Attention mechanism

| **Mechanism** | **BLEU** |
|---|---|
| Luong | 21.97 |
| Scaled Luong | 22.21 |
| Bahdanau | 23.78 |
| Normed Bahdanau (baseline) | **24.17** |

### 6.3.3. Attention Architecture

While the attention mechanisms differ more, the attention architectures are designed to accelerate the speed of the training. Here we test their impact on the translation quality.

As can be seen in table 6.10 the gnmt attention architecture v2 from Google improves the quality of the translation system. The attention architecture gnmt and gnmt v2 are created to speed up training, but because of the different calculation of the attention, the BLEU is affected. Unfortunately, due to limited resources, we are not able to train with the standard attention architecture. The standard attention architecture computes the attention based on the current top layer where as

in the gnmt architecture the attention is calculated based on the previous bottom layer. The second version of the gnmt architecture uses the current bottom layer to compute the attention. We believe that the improved BLEU is achieved due to the fact that the gnmt v2 architecture uses a more recent layer to compute BLEU. The best BLEU is again achieved on the baseline model, the BLEU over training steps are reported in figure 6.13.

Table 6.10.: Attention architecture

| **Architecture** | **BLEU** |
|---|---|
| gnmt | 23.67 |
| gnmt_v2 (baseline) | **24.17** |

### 6.3.4. Number of units

With a larger encoder and decoder dimensionality the system should be able to store more information and thus perform better. In order to validate this we decided to see how different dimensions impact the performance. Table 6.3.4 shows that a higher dimensionality improves the BLEU. As we can see, already with 256 units, we get a BLEU over 20, with 1152 hidden units the BLEU is about 6 points higher than with nine times less units (128). The best BLEU is achieved with the highest number of units as we had expected it. In figure 6.14 we plot the BLEU in relation to the number of units, we can see the BLEU first starts to increase more but starts to plateau with 640 units.

| Number of units | BLEU |
|---|---|
| 128 | 18.56 |
| 256 | 21.30 |
| 384 | 22.51 |
| 512 | 22.84 |
| 640 | 23.80 |
| 768 | 23.90 |
| 896 | 24.19 |
| 1024 (baseline) | 24.17 |
| 1152 | **24.38** |

Table 6.11.: BLEU scores for several number of units.

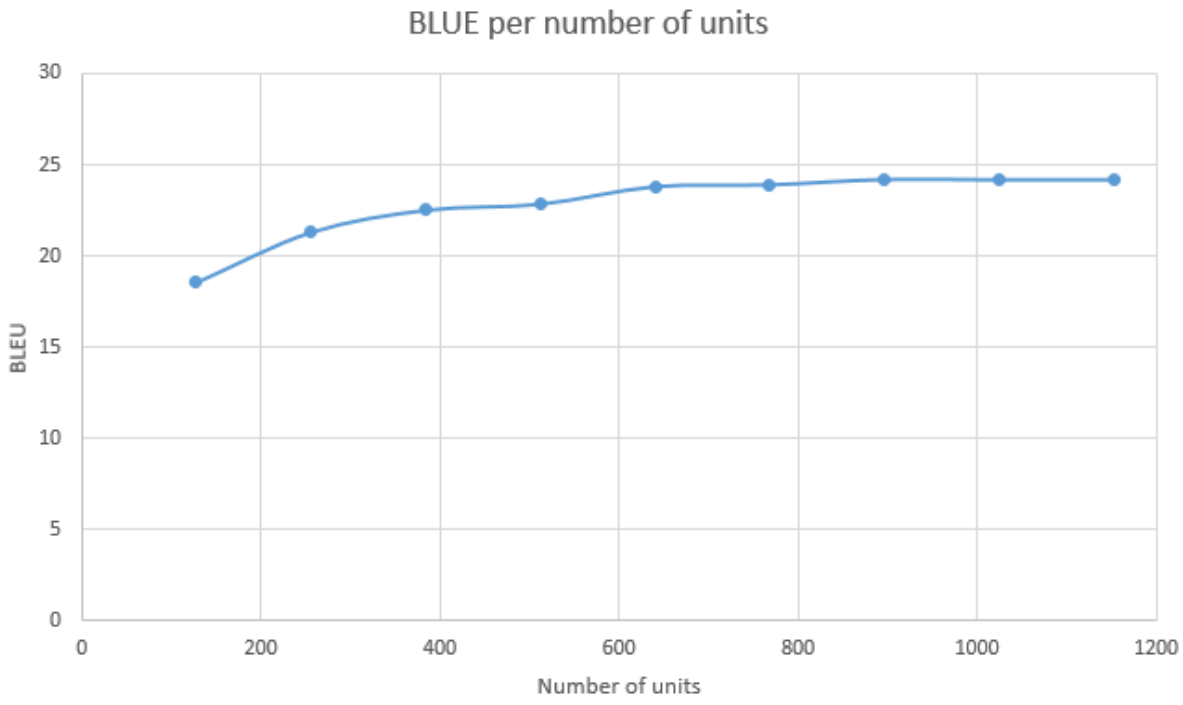


Figure 6.14.: BLEU over number of units.

### 6.3.5. Epoch

The number of epochs is a hyper-parameter, which decides if we waste computational resources or if our model continues to improve. In Figure 6.13 we plot the BLEU over 340'000 steps, the equivalent of approximately ten epochs. We can see that the BLEU plateaus after 230'000 steps or about 6.5 epochs. Figure 6.15 shows the loss during training. Unfortunately the validation loss is not calculated on the baseline model. This and the BLEU over time shows us, that we may still improve performance in terms of BLEU. However we believe that more than 20 Epochs would lead to overfitting.

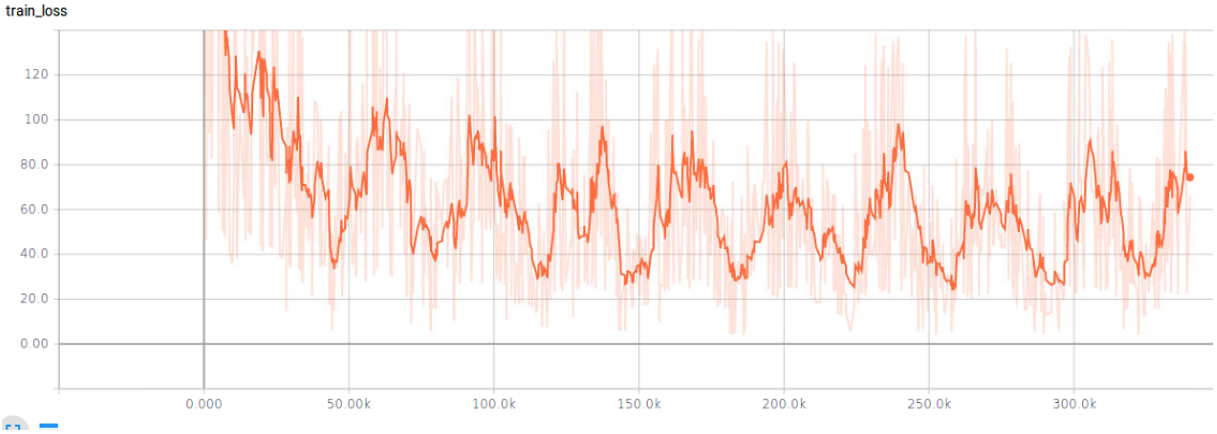


Figure 6.15.: Loss during training over steps of our baseline model. (Taken from tensorboard, x-axis=training steps, y-axis=train loss)

### 6.3.6. Beam width

Another hyper-parameter we are looking at is beam width. We expect that with a increasing beam width the model should produce better translations but the training should be slower due to the increased number of decoding steps required. As we see in table 6.3.6 a increasing beam width does improve the BLEU score. However, with a beam width of four the performance almost reaches the maximum. Increasing the beam width to eight only increases the BLEU only 0.1 point. In figure 6.16 we plot the BLEU per beam width.

| Beam width | BLEU |
|---|---|
| 0 | 23.51 |
| 4 | 24.14 |
| 8 | 24.22 |
| 10 (baseline) | 24.17 |
| 12 | 24.08 |
| 16 | **24.37** |

Table 6.12.: BLEU scores with different beam widths.

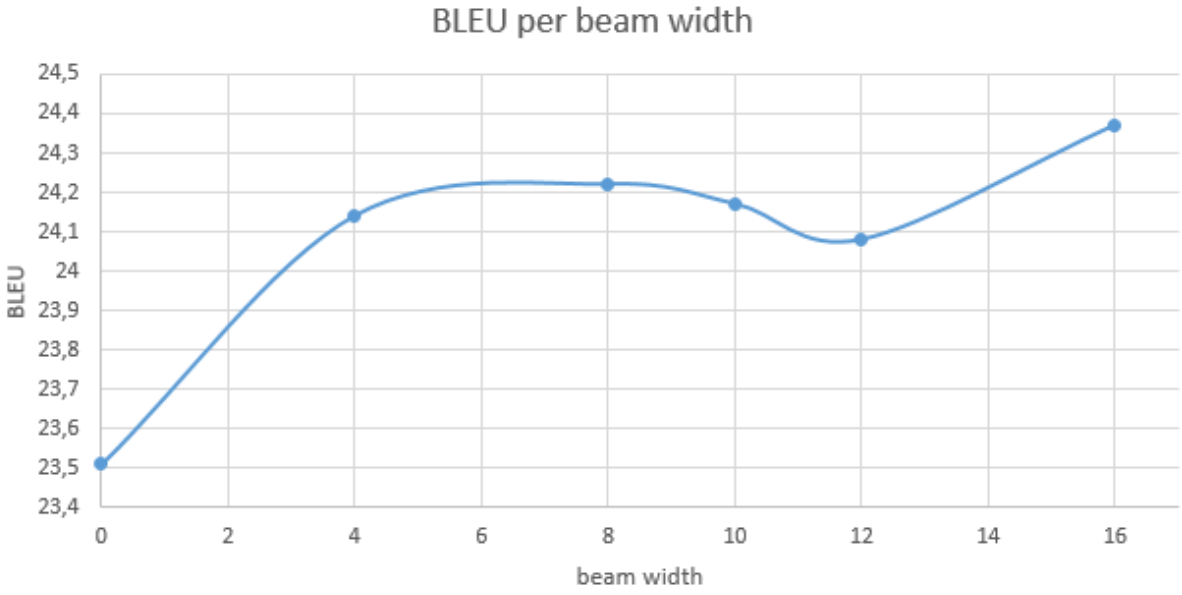


Figure 6.16.: BLEU over beam widths.

### 6.3.7. Number of layers

The number of layers in an NMT system is another parameter which seems to be important because a deeper model should be able to better capture information and produce translations than a shallower one. Here we investigated this hypothesis and as we can see in table 6.3.7 it is true for our tested model. However, a model with three or five layers performs worse than a model with only two layers. The model with 8 layers has the highest BLEU. Figure 6.17 shows the BLEU over time of this model.

Additionally in Figure 6.18 we plot the BLEU for every number of layers, there we can see that models with an low odd number of layers are worse than models with an even number of layers. This is a result of the model with an odd number of layers having one more layer in the encoder than in the decoder.

| **Number of layers** | **BLEU** |
|---|---|
| 2 | 23.41 |
| 3 | 20.16 |
| 4 (baseline) | 24.17 |
| 5 | 21.20 |
| 6 | 24.48 |
| 7 | 24.34 |
| 8 | **24.60** |

Table 6.13.: Number of layers

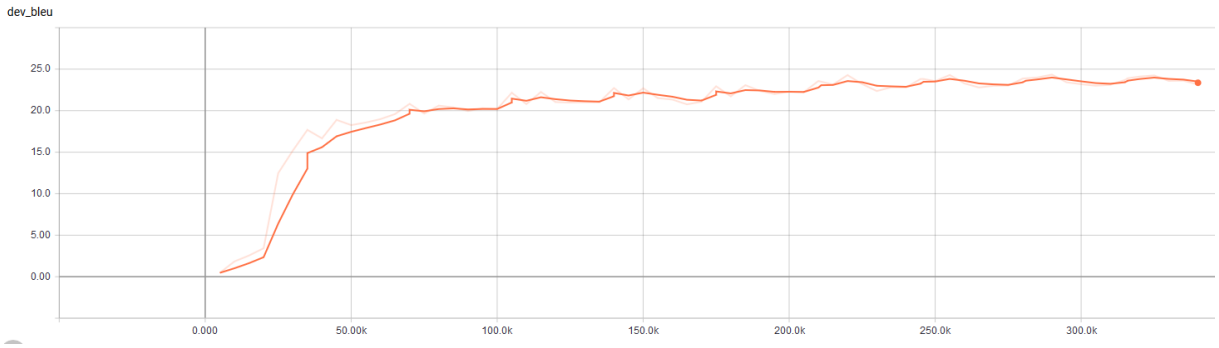


Figure 6.17.: BLEU over steps of the model with 8 layers. (Taken from tensorboard, x-axis=training steps, y-axis=BLEU)

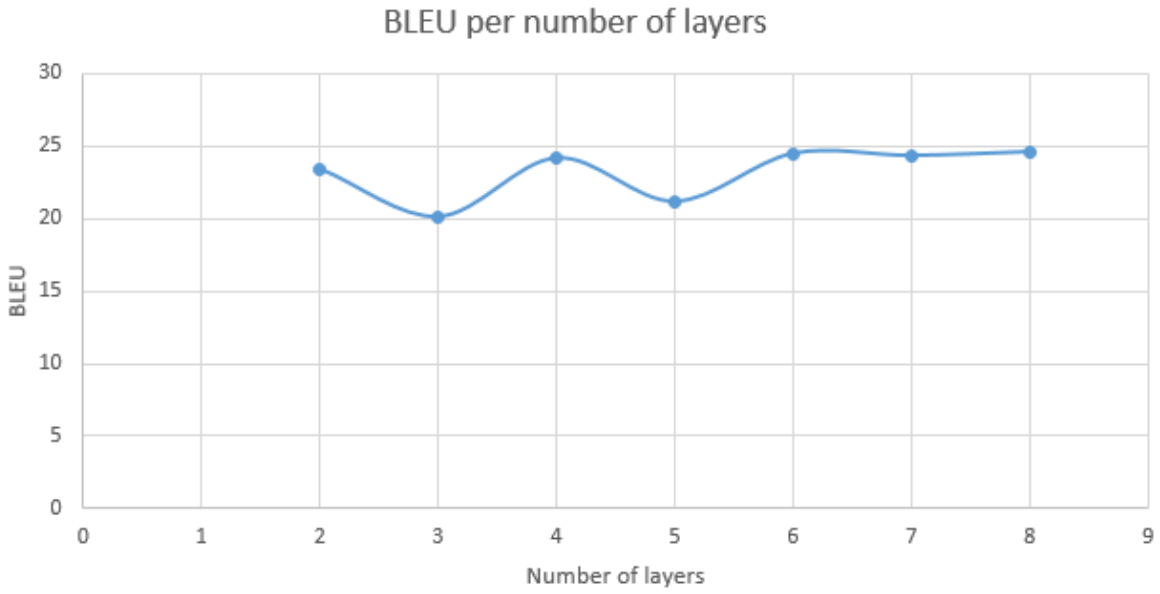


Figure 6.18.: Relation between number of layers and BLEU score.

## 6.4. Google's production system vs. DeepL's production system

As a last experiment we compared Google's and DeepL's production system. As we said in section 4.5 and 4.4 we used existing techniques to get the translations for the newstest2013 dataset. The results of this comparison are in table 6.4. As we expected, we can see that the system of DeepL is better than Google's system. However, both of them achieve state-of-the-art results with their production systems.

| System | BLEU |
|---|---|
| Google | 39.704 |
| DeepL | 42.530 |

Table 6.14.: Google and DeepL's production systems

# 7. Results

With our experiments in section 6.1 we compared different implementations of a NMT system on different datasets. Our aim was to familiarize ourselves with the machine learning library *Keras*, how to implement a seq2seq model and compare the performance of different implementations. Our findings from these experiments are that one of the biggest problems of the used implementations, is that the vocabulary of these implementations does not contain all words of the validation set, which leads to a worse BLEU score. To handle the problem of out-of-vocabulary words we propose to use a more advanced technique like BPE, which we explained in section 3.8 and was also in the Google nmt system section 4.4 and subsection 6.3.1. Additionally we find that larger datasets improve the BLEU and that more advanced techniques do not improve the BLEU on small datasets. Furthermore we found that on bigger datasets, techniques such as bidirectional layers helps to produce better translations. In all of our experiments in this section we obtained worse BLEU scores with the use of attention and unfortunately we were unable to find the reason for that. We suspect a bug exists in the implementation of the attention mechanism or in the use of the python package, which provides the attention mechanism, but were unable to verify this.

In the experiments of section 6.2 we try to gain a better understanding of how the encoder stores information about source sequences into the thought vector. We have encoded some sample source sentences into the thought vector with the best models of our experiments in section 6.1 and then we reduce the dimensionality of these vectors with PCA to plot them in three dimensions, to test if the encoder groups similar sentences into the same subspace. These plots show that

our models do not encode what we had expected. We think that the low BLEU score is a direct result of this. Papers by Cho [12], Sutslever [13] and Ming [51] related to this section showed in their work that the encoder should group similar sentences together. We were unable to reproduce this to an adequate degree.

In our last experiments in section 6.3 we investigate the impact of different hyper-parameters for a given baseline NMT model. To be more confident of our results we propose to do multiple runs per parameter setting and present the variance and standard deviation, to improve the statistical significance of the measured BLEU scores. We found that the normed Bahdanau attention mechanism gives the best BLEU and Bahdanau is in general better than the Luong attention mechanism. The Bahdanau attention mechanism is an additive one, which we found to be better, but the multiplicative Luong attention mechanism is computationally less expensive. The newest gnmt attention architecture v2 improves the BLEU a bit compared to v1. We also found that more hidden units improve the translation quality and we propose to test the performance of models with even more hidden units than we examined in this work. Unfortunately, do so requires more GPU memory. This can be achieved by panellizing the model onto multiple GPUs. We propose to train the model with more hidden units with a step size of 128 steps as we did in 6.14 and plot the BLEU in relation to unit size, until the curve starts to decrease. In testing several beam widths we found that a width of 16 gives us the best result, but compared to a beam width of eight it does not recieve and significantly larger improvement. Increasing the beam width leads to a decrease the training speed, as shown in table 6.3.6. To test if deeper networks are better than shallow networks for NMT we test different numbers of layers. We expected a increasing BLEU with deeper networks, because they should recognize complexer structures than shallow ones. As we showed in figure 6.18 models with odd numbers look to be worse than models with even numbers, we think the problem is

that the decoder is not as good in decoding as the encoder in encoding, because the encoder has more layers than the decoder and this leads to more power on the encoder. We propose to build a model with all the hyper-parameters which received the best BLEU, in order to verify if the BLEU of a joint model increases. Table 7 shows the joint hyper-parameters based on our exploration. As a final recommendation, if someone should want to start with building a state-of-the-art NMT system we propose to start with the Google seq2seq framework and use it as a base to build on. On the other hand, if you want to understand NMT in detail, we advise you to build your own system from scratch or to use the Google seq2seq framework and switch off every technique which is not necessary. From there you can start to experiment with a nearly vanilla NMT system and see which technique and how much it impacts the translation quality.

| **parameter** | **value** |
|---|---|
| attention mechanism | normed bahdanau |
| attention architecture | gnmt_v2 |
| training steps | 780’000 about 20 epochs |
| beam width | 16 |
| number of layers | 8 |

Table 7.1.: Best hyperparameters summary based on our exploration in section 6.3

# 8. Index

## 8.1. Glossary

**Activation Function** A *activation function* represents the mathematical function which belongs to a single *neuron*. 15

**anaconda** *Anaconda* is a software which makes it easier to manage python packages. 110

**encoding** A method which maps words to a matrix consisting of 1's and 0's. 24

**Hypothesis** In the context of machine translation a *hypothesis* refers to the output produced by the NMT, given an input sentence. 30

**Neuron** A *neuron* in the context of *NNs* represents a function. It's task is to receive a given number of input variables and produce a given number of output variables. Additionally the input variables have the option of being weighted. At a minimum they receive and produce one input variable. 15

**parallel corpora** A large amount of text translated accurately in two languages 13

**pip** *pip* is the package manager from python, to download and manage different python packages. 110

**recurrent neural network** A neural network in which connections between neurons form a directed circle. 13

**reference** A *reference* refers to a the correct translation against which the translation produced by the NMT can be measured. It is either an entire paragraph,

sentence or word. Traditionally a *reference* refers to one complete sentence.

17

## 8.2. List of Figures

## 8.3. List of Tables

# A. Appendix

## A.1. Technical Documentation

In the following section we will explain how to setup and use the implementation of the system we developed during this work.

### A.1.1. Prerequisites

To run the software several software requirements must be fulfilled and we recommend to have a decent hardware system with a GPU from Nvidia.
The experiments and the software itself could be run without a GPU but this will lead to a really bad performance and especially the training of the models will take a lot of time. We recommend to use python in the version 3.5.2.
If you want to use a GPU, you have to install the following software:

- Nvidia GPU driver for your GPU
- Nvidia cuda 9.0
- Nvidia cudnn 7.0

Additional the following python packages need to be installed. To avoid version compatibility issues and maybe different result we highly recommend to use the exact version of each package. All of this packages could be installed with the package manager of python *pip*. If you want to use anaconda you probably have to install some of these packages anyway with pip because some of this packages are not available in anaconda's package manager.

- keras 2.0.8
- tf-nightly-gpu if you use a GPU else tf-nightly, both in version 1.5.0-dev20171115

- gensim 3.1.0
- numpy 1.12.1
- matplotlib 2.0.2
- h5py 2.7.0

Additional to the python packages you need two more python packages, which you can get from github. Seq2seq framework for keras: https://github.com/farizrahman4u/seq2seq Complex seq2seq framework for keras: https://github.com/farizrahman4u/recurrentshop

### A.1.2. Repository Structure

In the following tables we show and describe the structure of the repository. Table A.1 shows the top level of the repository and table A.3 shows the content of the <repo>/NMT/src directory.

Table A.1.: Repository top structure

| **Directory** | **Content** |
|---|---|
| DataProcessingTools | Some tools for data processing. |
| DataSets | Here all the training/validation/test data and embeddings got stored. |
| eval_google_deepl_live | Contains the translations from the google and deepl production system and their tokenized form. |
| Evaluations | Here are the evaluations stored. |
| hparams | Here are the hyperparameters of all models are stored. |
| NMT | Here is the code of the software system. Detailed in table A.3. |
| Persistence | Here the model weightfiles and output of the training gots stored. |

Table A.2.: NMT/src directory structure

| Directory | Content |
|---|---|
| . | In the root directory are the main scripts, for training, evaluation and prediction. |
| data | In this directory are the scripts to preprocess and download the required data. |
| visualization | Here is the ipyhton notebook to visualize the hiddenstate. |
| google | In this directory is all the code to use googles nmt system. |
| helpers | Here are some helper scripts. |
| metrics | Here are the scripts to evaluate the BLEU score. |
| models | Here are all models but the google model. |

### A.1.3. Download

The software system itself could be downloaded from the github of the ZHAW using git or it is also on the USB-Stick. The data we used during this work could be downloaded or also be found on the USB-Stick, In the following we refer to the location where you saved the repository as <repo>. Copy all the content of the USB-Stick into <repo>.

#### WMT16 dataset

Run the following shell script. It will download and preprocess the required data for this dataset, you can find it in ”<repo>/NMT/src/data”.

wmt16_en_de.sh

#### WMT16 1 mio. dataset

To prepare this dataset you have to first prepare the WMT16 dataset as explained above in subsection A.1.3.

This preprocessing is based on the following ipython notebook you can find it in ”<repo>/NMT/src/data”.

EN2DE_fastai_preprocessing.ipynb

In cell three of this notebook you have to set the following variables:

NUM_TRAINING_SAMPLES = 1000000

DATASET_NAME = ’en_de_base_wmt16’

Then you can run all cells.

**Tatoeba dataset**

The data for this dataset is on the usb stick, to use it copy everything to your <repo> folder.

This preprocessing is based on the following ipythonnotebook.

EN2DE_fastai_preprocessing.ipynb

In cell three of this notebook you have to set the following variables:

NUM_TRAINING_SAMPLES = None

DATASET_NAME = ’en_de_base_tatoeba’

Then you can run all cells.

**Question dataset**

The data for this dataset is on the usb stick, to use it copy everything to your <repo> folder.

This preprocessing is already done, because this dataset is based on a large one, but only uses a very small subset.

If you want to preprocess it by yourself, you can get the data from http://www.statmt.org/wmt09/training-giga-fren.release2.tar. After downloading it, you have to unpack it and move it to the directory ”<repo>/DataSets/”. Then you can run all

cells in the following ipyhonnotebook. EN2FR_fastai_preprocessing.ipynb

### A.1.4. Scripts

In this subsection we explain how to use the provided scripts. We explain only the scripts, which are necessary for carrying out the experiments. All the following scripts can be found in ”<repo>/NMT/src/”.

Table A.3.: NMT/src directory structure

| **Name** | **Description** |
|---|---|
| create_hparams_google_nmt.py | With this script you can create the .json specification files for the experiments with google’s nmt system. |
| estimate_hiddenstates.py | With this script you can calculate the hiddenstate of each model, like we did in section [6.2]. |
| eval_everything.py | With this script you can calculate the BLEU score of each prediction file for each model in the directory <repo>/Persistence. |
| eval_google_and_deepl.py | With this script you can calculate the BLEU of the prediction file from google’s and deepl’s production system. |
| get_max_bleu_score.py | This scripts retrieves and prints the max BLEU of each model. |
| run_eval.py | With this script you can evaluate the BLEU of each model individually. |
| run_prediction.py | With this script you can use/test the models, you can predict from file or within the console. |
| run_training.py | This script is used to train the different models. |
| stat_bleu.py | This script writes the BLEU of the models, for each epoch into a separate .csv file. |
| translate_with_deepl.py | This script uses the API of DeepL to translate each paragraph of the validation and test set with. |

### A.1.5. Conducting the Experiments

All experiments could be done through python scripts, how to do this is explained here. For the experiments in section 6.1 and 6.3 we used configuration files in JSON format, these format is explained below. The format of the experiments in section 6.1 and section 6.3 are different. Table A.4 shows the format of the experiments in section 6.1, the config files are in <repo>/hparams.
Table A.5 shows the format of the experiments in section 6.3, the config files are in <repo>/our_hparams.
6.2 To make this experiments, you have to run estimate_hiddenstate.py, it will produce a .txt file with the hiddenstates for the given input sentences. In the script you can choose another NMT model and the input sentences.

After creating the hiddenstates you have to start the ipyhthon notebook PCA_plot_-sentences.ipynb, which you can find in <repo>/NMT/src/general/visualization In the fourth cell you can specify the path of the file which contains the hiddenstates.

### A.1.6. Conducting Inference

The inference with each model could be done via one python script. As described in table A.3 the run_prediction.py script is used to do inference.
When the script is executed, it will first ask which model you want to use for prediction, you can either type in the name or the number of the model. Then you can choose the submodel like with attention or without. After you choosed the submodel you can choose between interactive sentence prediction or batch prediction from file. If you choose prediction per sentence, you can just type a sentence and hit enter. If you choose batch prediction you have to type in the source file and then the output file.

| **Parameter name** | **description** |
|---|---|
| dim_src_lang_vec | Dimension of the embeddings of the source language. |
| dim_tgt_lang_vec | Dimension of the embeddings of the target language. |
| maxlen | Max length of the input and output sequences. |
| n_en_vec | How many words are in the english embeddings. |
| lr | The learning rate of the model. |
| batch_size | The batch size for training the model. |
| epochs | How many epochs the model should be trained for. |
| optimizer | The optimizer of the model. |
| hidden_dim | How big the hidden dimension of each recurrent cell should be. |
| attention | Boolean flag, which states, if attention is used or not. |
| sos | Flag which indicates if the start-of-sequence token is used. |
| eos | Flag which indicates if the end-of-sequence token is used. |
| unk | Flag which indicates if the unknown token is used. |
| num_training_samples | Number of training samples, which are used, if null, all are used |
| expected_src_vocab_size | Max size of the source vocabulary. |
| expected_tgt_vocab_size | Max size of the target vocabluary. |
| dataset_name | Name of the dataset, which should be used. |
| p_dense_dropout | The dropout rate, if dropout is used |

Table A.4.: Description of the different configuration parameters for the compare different implementations experiment section.

| Parameter name | description |
|---|---|
| attention | Which attention mechanism should be used. |
| attention_architecture | The attention architecture which should be used. |
| batch_size | The training batch size. |
| colocate_gradients_with_ops | Flag if the gradients should be colocate with the operations to speedup training. |
| decay_factor | The decay factor of the learning rate. |
| decay_steps | After how many steps the decay should be start. |
| dropout | The dropout rate. |
| encoder_type | The type of the encoder cells. |
| eos | Indicates which token stands for the end-of-sequence token. |
| forget_bias | The initial forget bias of the recurrent cells, if they have a forget gate. |
| infer_batch_size | Batch size during inference. |
| init_weight | Initial weight value. |
| learning_rate | The initial learning rate. |
| max_gradient_norm | Max gradient norm, for clipping the gradients. |
| metrics | The evaluation metrics. |
| num_buckets | Number of buckets. |
| num_layers | How many layers the model should have. |
| num_train_steps | Number of training steps, where about 34'000 are one epoch at a batch size of 128. |
| num_units | Number of units for the recurrent cells. |
| optimizer | The optimizer. |
| residual | If residual layers should be used. |
| share_vocab | Flag, if the source and target vocabulary are shared. |
| subword_option | Which subword encoding should be used. |
| sos | Which token is the start-of-sequence token. |
| source_reverse | If the source sentence should be reversed. |

Table A.5.: Description of the different configuration parameters for the parameter exploration experiment section.

| **Parameter name** | **description** |
|---|---|
| src_max_len | Max length of the source sequence during training. |
| src_max_len_infer | Max length of the source sequence during inference. |
| start_decay_step | At which step the learning rate decay starts. |
| steps_per_external_eval | How many steps external evaluation should be done. |
| steps_per_stats | How many steps per statistic should be done. |
| tgt_max_len | Max length of the target sequence during training. |
| tgt_max_len_infer | Max length of the target sequence during inference. |
| time_major | If the model should be time major. |
| unit_type | The type of the recurrent units. |
| beam_width | The beam width value. |
| length_penalty_weight | Initial value of the length penalty weight. |
| src | The source language e.g. en for english. |
| tgt | The target language e.g. de for german. |

Table A.6.: Description of the different configuration parameters for the parameter exploration experiment section.

# Bibliography

[1] U. P. Union, *History*, 2016. [Online]. Available: http://www.upu.int/en/the-upu/history/about-history.html (visited on 11/18/2017).

[2] B.-K. Kim, *Internationalizing the Internet : the co-evolution of influence and technology*. Edward Elgar Pub, 2005, p. 300, ISBN: 9781843764977.

[3] B. Sanou, *Youth are at the Forefront of the internet adoption*, Geneva, 2017. [Online]. Available: https://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2017.pdf.

[4] S. R. Anderson, *How many languages are there in the world?* 2010. [Online]. Available: https://www.linguisticsociety.org/content/how-many-languages-are-there-world (visited on 11/18/2017).

[5] B. Turovsky, *Ten years of google translate*, 2016. [Online]. Available: https://blog.google/products/translate/ten-years-of-google-translate/.

[6] N. Woolf, *Google Translate update seeks to break language barrier | Technology | The Guardian*, 2015. [Online]. Available: https://www.theguardian.com/technology/2015/jan/14/google-translate-app-update-break-language-barrier (visited on 12/10/2017).

[7] M. Popović, "The Prague Bulletin of Mathematical Linguistics Comparing Language Related Issues for NMT and PBMT between German and English," 2017. DOI: 10.1515/pralin-2017-0021. [Online]. Available: https://www.degruyter.com/downloadpdf/j/pralin.2017.108.issue-1/pralin-2017-0021/pralin-2017-0021.pdf.

[8] J. Hutchins, "First Steps in Mechanical Translation," [Online]. Available: https://pdfs.semanticscholar.org/649e/ef8de42d86f840b4e2e5913deadb8bc1ea29.pdf.

[9] European Parliament - The Secretary-General, *Directorate-General for Translation | The Secretary-General | European Parliament*. [Online]. Available: http://www.europarl.europa.eu/the-secretary-general/en/organisation/directorate-general-for-translation (visited on 12/23/2017).

[10] I. Thompson, *Swedish | About World Languages*, 2015. [Online]. Available: http://aboutworldlanguages.com/swedish (visited on 01/07/2018).

[11] Pangeanic, *What is The Size of the Translation Industry? - Pangeanic Translations*. [Online]. Available: https://www.pangeanic.com/knowledge{_}center/size-of-the-translation-industry/ (visited on 01/07/2018).

[12] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," 2014, [Online]. Available: http://arxiv.org/abs/1406.1078.

[13] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," 2014. arXiv: arXiv:1409.3215v3. [Online]. Available: https://arxiv.org/pdf/1409.3215.pdf.

[14] Microsoft, *Translate text and voice calls | Skype Translator | Skype*. [Online]. Available: https://www.skype.com/en/features/skype-translator/ (visited on 12/23/2017).

[15] P. Koehn and R. Knowles, "Six Challenges for Neural Machine Translation," pp. 28–39, 2017. [Online]. Available: http://www.aclweb.org/anthology/W/W17/W17-3204.pdf.

[16] M. Hutson, "Computer chip mimics human brain, with light beams for neurons," *Science*, 2017. DOI: 10.1126/science.aan6998. [Online]. Available: http://www.sciencemag.org/news/2017/06/computer-chip-mimics-human-brain-light-beams-neurons.

[17] S. W. Smith, *Neural Network Architecture*. [Online]. Available: http://www.dspguide.com/ch26/2.htm (visited on 12/12/2017).

[18] V. Valkov, *Creating a Neural Network from Scratch - TensorFlow for Hackers (Part IV)*, 2017. [Online]. Available: https://medium.com/@curiousily/tensorflow-for-hackers-part-iv-neural-network-from-scratch-1a4f504dfa8 (visited on 12/11/2017).

[19] M. A. Ranzato, S. Chopra, M. Auli, and W. Zaremba, "SEQUENCE LEVEL TRAINING WITH RECURRENT NEURAL NETWORKS," [Online]. Available: https://arxiv.org/pdf/1511.06732.pdf.

[20] M. Norouzi, S. Bengio, Z. Chen, N. Jaitly, M. Schuster, Y. Wu, and D. Schuurmans, "Reward Augmented Maximum Likelihood for Neural Structured Prediction," 2017. arXiv: arXiv:1609.00150v3. [Online]. Available: https://arxiv.org/pdf/1609.00150.pdf.

[21] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Ł. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," [Online]. Available: https://arxiv.org/pdf/1609.08144.pdf.

[22] R Rojas, "The Backpropagation Algorithm," *Neural Networks*, 1996. [Online]. Available: https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf.

[23] R. Socher and S. University, "CS224n: Natural Language Processing with Deep Learning," 2017. [Online]. Available: http://web.stanford.edu/class/cs224n/.

[24] P. Radhakrishnan, *Introduction to Recurrent Neural Network*, 2017. [Online]. Available: https://towardsdatascience.com/introduction-to-recurrent-neural-network-27202c3945f3 (visited on 12/11/2017).

[25] C. Olah, *Understanding LSTM Networks*, 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 12/11/2017).

[26] Y. Bengio, P. Simmard, and Fransconi Paolo, "Learning Long-Term Dependencies with Gradient Descent is Difficult," *IEEE Transactions On Neural Networks*, vol. 5, p. 157, 1994. [Online]. Available: http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf.

[27] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," 2012. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.421.8930{\&}rep=rep1{\&}type=pdf.

[28] S. Hochreiter and S. Jurgen, "LONG SHORT-TERM MEMORY," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: http://www.bioinf.jku.at/publications/older/2604.pdf.

[29] S. Robertson, *Practical PyTorch: Translation with a Sequence to Sequence Network and Attention*. [Online]. Available: https://github.com/spro/practical-pytorch/blob/master/seq2seq-translation/seq2seq-translation.ipynb (visited on 12/11/2017).

[30] D. Bahdanau, K. Cho, and Y. Bengio, "NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE," [Online]. Available: https://arxiv.org/pdf/1409.0473.pdf.

[31] M.-T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," 2015. arXiv: arXiv:1508.04025v5. [Online]. Available: https://arxiv.org/pdf/1508.04025.pdf.

[32] T. Salimans and D. P. Kingma, “Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks,” [Online]. Available: https://arxiv.org/pdf/1602.07868.pdf.

[33] M.-T. Luong, E. Brevdo, and R. Zhao, “Neural Machine Translation (seq2seq) Tutorial,” *https://github.com/tensorflow/nmt*, 2017. [Online]. Available: https://github.com/tensorflow/nmt.

[34] A. Choudhary, *What is teacher forcing in RNN?* 2017. [Online]. Available: http://www.cedar.buffalo.edu/{~}srihari/CSE676/10.2.1TeacherForcing.pdf (visited on 12/09/2017).

[35] E. Denoual and Y. Lepage, “BLEU in characters: towards automatic MT evaluation in languages without word delimiters,” 2005. [Online]. Available: http://www.mt-archive.info/IJCNLP-2005-Denoual.pdf.

[36] Merriam-Webster, *How many words are there in English?* [Online]. Available: https://www.merriam-webster.com/help/faq-how-many-english-words (visited on 12/31/2017).

[37] R. Sennrich, B. Haddow, and A. Birch, “Neural Machine Translation of Rare Words with Subword Units,” 2016. arXiv: arXiv:1508.07909v5. [Online]. Available: https://arxiv.org/pdf/1508.07909.pdf.

[38] P. Gage, “A New Algorithm for Data Compression,” *C Users J.*, vol. 12, no. 2, pp. 23–38, 1994, ISSN: 0898-9788. [Online]. Available: http://dl.acm.org/citation.cfm?id=177910.177914.

[39] K. Ekštein and V. Matoušek, Eds., *Text, Speech, and Dialogue*, ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, vol. 10415, ISBN: 978-3-319-64205-5. DOI: 10.1007/978-3-319-64206-2. [Online]. Available: http://link.springer.com/10.1007/978-3-319-64206-2.

[40] L. Derksen, *Visualising high-dimensional datasets using PCA and t-SNE in Python*, 2016. [Online]. Available: https://medium.com/@luckylwk/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python-8ef87e7915b (visited on 12/31/2017).

[41] L. Van Der Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008. [Online]. Available: http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf.

[42] S. Witteveen, *Talk05 Seq2Seq NMT*, 2017. [Online]. Available: https://github.com/samwit/TensorFlowTalks/tree/master/Talk05{_}Seq2Seq{_}NMT (visited on 01/06/2017).

[43] Gooogle, *Neural Machine Translation (seq2seq) Tutorial*, 2017. [Online]. Available: https://github.com/tensorflow/nmt (visited on 01/06/2018).

[44] T. Luong and E. Brevdo, *Building Your Own Neural Machine Translation System in TensorFlow*, 2017. [Online]. Available: https://research.googleblog.com/2017/07/building-your-own-neural-machine.html (visited on 01/06/2018).

[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf.

[46] DeepL GmbH, "Press Release: DeepL," Tech. Rep., 2017. [Online]. Available: https://www.deepl.com/press.html.

[47] Keras, *Keras: The Python Deep Learning library*. [Online]. Available: https://keras.io/ (visited on 01/06/2018).

[48] moses smt, *mosesdecoder*, 2017. [Online]. Available: https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/tokenizer.perl (visited on 01/07/2018).

[49] R. S. Sennrich, *subword-nmt*, 2017. [Online]. Available: https://github.com/rsennrich/subword-nmt (visited on 01/07/2018).

[50] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention Is All You Need,” [Online]. Available: https://arxiv.org/pdf/1706.03762.pdf.

[51] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, Y. Song, and H. Qu, “Understanding Hidden Memories of Recurrent Neural Networks,” [Online]. Available: https://arxiv.org/pdf/1710.10777.pdf.