An Exploration of Alignment Concepts to Bridge the Gap between Phrase-based and Neural Machine Translation

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der
RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker
Jan-Thorsten Peter
aus Soltau

Berichter: Universitätsprofessor Dr.-Ing. Hermann Ney
Universitätsprofessor Dr. Josef van Genabith

Tag der mündlichen Prüfung: 22. July 2020

"Seeking what is true is not seeking what is desirable."

— *Albert Camus, The Myth of Sisyphus and Other Essays*

Acknowledgments

Many impressive and intelligent persons assisted me in producing this thesis. I would like to express my deepest thanks and appreciation to each and every one of them.

Prof. Dr.-Ing. Hermann Ney offered me the opportunity to work on my doctorate within the Human Language Technology and Pattern Recognition Group. The resources and possibilities attached to this position were crucial and allowed me to connect to many talented people. I was very pleased that Prof. Dr. Josef van Genabith accepted to be my second supervisor, and showed great interest in my work.

Before starting my work as a Ph.D. student, Daviid Vilar introduced me to machine translation, zsh, and vim. Daniel Stein ("ergibt Sinn") and Matthias Huck had enough patience to supervise and correct my Diploma Thesis. Thank you for that and all the good times we had at the conferences. Many thanks also to Arne Mauser and Thomas Deselaers for introducing me to data mining and neural networks.

I want to thank my office-mates and friends Andy, Malte, and Jan for enjoying many useful and many useless discussions in and outside the office; My sysadmin colleagues: Stefan Koltermann, Thomas Dackweiler, Kai Frantzen, Weiyue Wang, Jan Rosendahl, Christian Plahl, David Nolden, and David Rybach; Pavel Golik, for consistently reminding me that it is, most of the time, not a good idea to change a running system; Secretaries Steffi, Anna Eva, Andrea, Dhenya, and Katja for taking care of all the administration work and for the short fun conversations in between.

Furthermore, I would like to thank Eugen, for being the first person to switch his workflow to Sisyphus and to support its development, Jörn Wübker, Stephan Peitz, and Markus Freitag, for taking on the senior role and showing me how things are done at the institute, project meetings, and conferences, Saab Mansour, for always finding a great bar, Tamer Alkhouli, for showing me that you do not have to drink at that bar to have fun, Patrick Dötsch, for helping me with many Theano problems, Albert Zeyer, for showing me CCC and keeping me fit, Parnia Bahar, for all the little jokes and conversation, Markus Nußbaum-Thom, for taking me skiing and all the other fun we had, and Oscar Koller, for helping me to optimize the time it takes to get to the train station.

Moreover, thanks to Weiyue Wang, Jan Rosendahl, Parnia Bahar, Malte Nuhn, Andreas Guta, Tamer Alkhouli, and Deanna Dorsett for proofreading this thesis. Also to all other people I have been privileged to work with including Jens, Yunsu, Tobias, Simon, Zoltán, Minwei, Muhammad, Christian O., Yannick, Basha, Christoph, Julian, Martin, Kazuki, Markus K., Willi, Mahdi, Mirko, Harald, Michal, Farzad, Ilya, and Amr. A special thanks also to my bachelor and master students Weiyue Wang, Nick Rossenbach, Arne Nix, Hendrik Rosendahl, and Pavel Petrushkov.

My junior year in high school was spent in California as an exchange student, hosted by Dave and Deanna Dorsett. My English skills to this day still leave room for improvement, but my time with them allowed me to gain confidence to function in an English-speaking world. Many long conversations especially with Dave helped me a great deal.

Finally, deepest thanks to my family. I am very grateful for the support, motivation, and love from my parents and grandparents that I received growing up and during this project, especially from my father. Most of all, I thank my wife, Tanja, and my children. Tanja supported me in every possible way to finish this thesis. My children, on the other hand, probably did not speed up this process, but they were always there to lighten up my day.

Abstract

Machine translation, the task of automatically translating text from one natural language into another, has seen massive changes in recent years. After phrase-based systems represented the state of the art for over a decade, advancements were made in the structure of neural networks and computational power. These advancements made it possible to build neural machine translation systems which first improved and later outperformed phrase-based systems.

These two approaches have their strength in different areas. The well-known phrase-based systems allow fast translations on CPU that can easily be explained by examining the translation table. In contrast, neural machine translation produces more fluent translations and is more robust to small changes in the provided input. This thesis aims to improve both systems by combining their advantages.

The first part of this thesis focuses on investigating the integration of feed-forward neural models into phrase-based systems. Small changes in the input of a phrase-based system can turn an event that was seen in the training data into an unseen event. Neural network models are by design able to handle such cases due to the continuous space representation of the input, whereas phrase-based systems are forced to fall back to shorter phrases. This means a loss of knowledge about the local context which results in a degradation of the translation quality. We combine the flexibility provided by feed-forward neural networks with phrase-based systems while gaining a significant improvement over the phrase-based baseline systems. We use feed-forward networks since they are conceptually simple and computationally fast.

Commonly, their structure only utilizes local source and target context. Due to this structure, they cannot capture long-distance dependencies. We improve the performance of feed-forward neural networks by efficiently incorporating long-distance dependencies into their structure by using a bag-of-words input.

The second part of this thesis focuses on the pure neural machine translation approach using the encoder-decoder model with an attention mechanism. This mechanism corresponds indirectly to a soft alignment. At each translation step, this model relies only on its previous internal state and the current decoder position to compute the attention weights. There is no direct feedback from the previously used attention. Inspired by hidden Markov models where the prediction of the currently-aligned position depends also on the previously-aligned position, we improve the attention model by adding direct feedback from previously-used attention to improve the overall model performance.

Additionally, we utilize word alignments for neural networks to guide the neural network during training. By incorporating the alignment as an additional cost function, the network performs better as our experiments show.

Even though the state-of-the-art neural models do not require word alignments anymore, there are still applications that benefit from good alignments. These include the visualization of parallel sentences, the creation of dictionaries, the automatic segmentation of long parallel sentences and

the above-mentioned usage during neural network training. We present a way to apply neural models to create word alignments that improve over word alignments trained with IBM and hidden Markov models.

These techniques are evaluated on various large-scale translation tasks of public-evaluation campaigns. Applying new methods with usually complex workflows to new translation tasks is a cumbersome and error-prone exercise. We present a workflow manager, which is developed as part of this thesis to simplify this task and enable an easier knowledge transfer.

Kurzfassung

Die maschinelle Übersetzung, das heißt die automatischen Übersetzung von Text von einer natürlichen Sprache in eine andere, hat in den letzten Jahren massive Veränderungen erfahren. Phrasenbasierte Systeme stellten mehr als ein Jahrzehnt lang den Stand der Technik da. Fortschritte in der Struktur neuronaler Netze und in der Rechenleistung ermöglichten den Aufbau neuronaler maschineller Übersetzungssysteme, welche zunächst die phrasenbasierten Systeme verbesserten und später übertrafen.

Die beiden Ansätze haben ihre Stärken in verschiedenen Bereichen. Die phrasenbasierten Systeme ermöglichen schnelle Übersetzungen auf der CPU und können durch Analyse der Übersetzungstabellen nachvollzogen werden. Im Gegensatz dazu erzeugt die neuronale maschinelle Übersetzung flüssigere Übersetzungen und ist robuster gegenüber kleinen Änderungen in den Eingabesätzen. Diese Dissertation zielt darauf ab, beide Systeme durch die Kombination ihrer Vorteile zu verbessern.

Der erste Teil dieser Arbeit konzentriert sich auf die Analyse und Integration neuronaler feedforward-Netze in phrasenbasierte Systeme. Schon kleine Änderungen in der Eingabe eines phrasenbasierten Systems können ein Ereignis, welches gesehen wurde, in ein ungesehenes Ereignis verwandeln. Neuronale Netze sind aufgrund der stetige Darstellung des Eingabesatzes im mehrdimensionalen im Vektorraum in der Lage, direkt mit solchen Fälle umzugehen. Phrasenbasierte Systeme sind hingegen gezwungen, auf kürzere Phrasen zurückzugreifen. Dies bedeutet einen Verminderung des Übersetzungskontextes, was im Allgemeinen zu einer Verschlechterung der Übersetzungsqualität führt. Wir kombinieren die Flexibilität, die durch neuronale feedforward-Netze geboten wird, mit phrasenbasierten Systemen um eine Verbesserung gegenüber den rein phrasenbasierten Systemen zu erzielen. Wir verwenden hier neuronale feedforward-Netze, da sie konzeptionell einfach anzuwenden sind.

In der Regel nutzen neuronale feedforward-Modelle nur einen lokalen Quell- und Zielkontext. Aufgrund dieser Struktur können sie keine Abhängigkeiten zwischen Wörtern über große Entfernungen erfassen. Wir verbessern die Leistung von diesen Modellen, indem wir die Abhängigkeiten über große Entfernungen effizient in ihre Struktur einbauen. Dafür nutzen wir einen Bag-of-Words Ansatz als zusätzlich Eingabe.

Der zweite Teil der Arbeit konzentriert sich auf den rein neuronalen maschinellen Übersetzungsansatz unter Verwendung des Encoder-Decoder-Modells in Verbindung mit einem Aufmerksamkeits-Mechanismus. Dieser Mechanismus entspricht indirekt der weichen Alignierung von Quell- zu Zielwörtern. Bei jedem Übersetzungsschritt verlässt sich dieses Modell nur auf seinen vorherigen internen Zustand und die aktuelle Decoder-Position, um die Aufmerksamkeitsgewichte zu berechnen. Es gibt kein direktes Feedback von den zuvor verwendeten Aufmerksamkeitsgewichten. Inspiriert durch das Hidden-Markov-Modell, bei welchem die Vorhersage der aktuellen Position von der vorherigen Position abhängt, verbessern wir das Aufmerksamkeitsmodell, indem wir eine direkte Verbindung zu den zuvor verwendeten Aufmerksamkeitgewichten hinzufügen.

Zusätzlich verwenden wir Wortalignierungen für neuronale Netze, um die neuronale Netze während des Trainings in die richtige Richtung zu lenken. Indem wir ein Abweichen des Aufmerksamkeits-Mechanismus vom Wort-Alignment als zusätzliche Kostenfunktion einbeziehen, erzielen so trainierte Netzwerke bessere Ergebnisse.

Auch wenn die neuronalen Modelle auf dem neuesten Stand der Technik keine Wortalignierungen mehr erfordern, gibt es immer noch Anwendungen, die von Alignments profitieren, wie zum Beispiel die Visualisierung paralleler Sätze, das Erstellen von Wörterbüchern, die automatische Segmentierung langer paralleler Sätze und das erwähnte Training neuronaler Netzen. Wir stellen eine Möglichkeit vor, neuronale Modelle anzuwenden, um Wortalignierungen zu erstellen, welche eine höhere Genauigkeit besitzen als Wortalignierungen, welche mit IBM- und Hidden-Markov-Modellen erzeugt wurden.

Wir evaluieren diese Techniken im Rahmen von mehreren öffentlichen Evaluierungskampagnen. Methoden, welche häufig komplexe Arbeitsabläufe mit sich bringen, auf neue Übersetzungsaufgaben anzuwenden ist eine mühsame und fehleranfällige Aufgabe. Wir stellen einen Workflow-Manager vor, welcher im Rahmen dieser Dissertation entwickelt worden ist, um diese Aufgabe zu vereinfachen und einen leichteren Wissenstransfer zu ermöglichen.

Contents

1. Introduction

Modern technology potentially allows today to communicate instantly with nearly everyone on the planet. With approximately 93% of the world population living within reach of a mobile broadband service [International Telecommunication Union 19], one of the most significant challenges is the language barrier. There are 286 distinct languages in Europe [Lewis & Simons$^+$ 15], and the European Union alone has, as of 2020, 24 official languages. At the same time, only a little over half the population in the European Union speaks more than one language [European Commission 12].

This shows the apparent need to translate between these languages. Generating human translations is, in practice, too expensive or too slow for many use cases. Machine translation systems currently still show a drop in quality compared to human translations. However, the heavy usage of online machine translation services shows that there is a high demand for them.

1.1 Machine Translation

The field of machine translation aims to translate from a human language into another human language, usually without human interaction. First attempts to use computers to automate translation started in the 1950s [Bar-Hillel 51, IBM 54]. These early systems were based on using handcrafted rules and dictionaries to translate. They showed promising results on academic tasks with well-formed input texts but also substantial weaknesses with real-world inputs that often do not follow all grammatical rules.

IBM showed that using statistically trained models was an alternative way to automatically generate translations [Brown & Cocke$^+$ 88, Brown & Cocke$^+$ 90]. These models computed, on a word level, probabilities for possible translations and could also handle sentences that could not be parsed using rule-based systems.

This approach was later extended to extract the translation of multiple words, so called phrases, from the training data and use these to translate sentences [Och & Tillmann$^+$ 99, Koehn & Och$^+$ 03]. These phrases allowed the system to take more context into account when computing the probabilities, which gave improvements in terms of translation quality. Even though phrase-based systems are able to capture short-range dependencies very well, they cannot capture dependencies that exceed the used phrase length. A phrase-based system that should translate a phrase, which was not seen in the training data, has to fall back to use smaller phrases or even words. This means that even small changes in the input can cause the phrase-based approach to lose its advantage over the word-based approach.

1.2 Neural Machine Translation

The history of artificial neural networks reaches back to 1943 [McCulloch & Pitts 43] when first attempts to simulate the human brain activity were made using electrical circuits. In the 1950s,

neural networks were simulated on computers [A. Clark & G. Farley 55, Rochester & Holland$^+$ 56], and showed some useful results, but it was still unclear how larger networks should be trained. The idea that backpropagation could be used to train neural networks was introduced in the 1970s [Werbos 75]. In 1986 [Rumelhart & Hinton$^+$ 86] showed that this method could be used to create useful internal representations of the given inputs. After these initial advances, neural networks dropped out of focus in most of the scientific community. The model training on CPUs took a long time compared to other machine learning methods.

Faster CPUs, and later even faster GPUs, combined with better tools, led to a comeback of neural networks in machine learning. The grown computation power allowed for larger networks, which combined with better network structures like LSTM [Hochreiter & Schmidhuber 97] and attention mechanism, allowing for larger and stronger models, which strongly improved state of the art in many fields.

This renaissance of neural networks in recent years allowed for a natural way to model these longer distance dependencies as well as allowing for a better generalization from seen events to unseen events. Neural machine translation [Bahdanau & Cho$^+$ 15] started to show competitive results to phrase-based systems in 2015 [Bojar & Chatterjee$^+$ 15] and continued to dominate the field after that.

1.3 About this Thesis

In general, the neural machine translation produce more fluent translations than the phrase-based approach, but it loses the property of being easily explainable by examining the translation table. We show in this thesis ways how to bridge the gap between phrase-based machine translation and neural machine translation from both directions.

In Chapter 2, we present the scientific goals of this thesis. Chapter 3 explains the methods in statistical machine translation used in this thesis. How neural networks work and how they can be trained is explained in Chapter 4.

In Chapter 5, we present methods on how to integrate some advantages of neural networks into phrase-based systems by including feed-forward neural networks into them. We also improve the capability of feed-forward neural networks to handle long-distance dependencies. The corresponding experiments are presented in Chapter 6.

In the following Chapter 7, we explain neural machine translation in detail, before presenting the methods we applied to improve it. Inspired by the word alignment used in statistical machine translation, we try to introduce these concepts into neural machine translation. Chapter 8 presents the experimental evaluation of these methods.

As part of this thesis, we developed a framework to manage the building of our systems, including training models and evaluating them. It is presented in Chapter 9. This thesis concludes with listing the scientific achievements in Chapter 10 and the individual contributions in Chapter 11.

1.4 Publications

The following scientific publications have been published at peer-reviewed conferences during the course of this thesis:

- Feed-forward Neural Networks in Statistical Machine Translation
  - A Comparison between Count and Neural Network Models Based on Joint Translation and Reordering Sequences [Guta & Alkhouli$^+$ 15]

  - Local System Voting Feature for Machine Translation System Combination [Freitag & Peter$^+$ 15]
  - Exponentially Decaying Bag-of-Words Input Features for Feed-Forward Neural Network in Statistical Machine Translation [Peter & Wang$^+$ 16]

- Neural Machine Translation Improvements
  - Alignment-Based Neural Machine Translation [Alkhouli & Bretschner$^+$ 16]
  - Generating Alignments Using Target Foresight in Attention-Based Neural Machine Translation [Peter & Nix$^+$ 17]
  - Empirical Investigation of Optimization Algorithms in Neural Machine Translation [Bahar & Alkhouli$^+$ 17]

- Toolkits
  - Jane 2: Open Source Phrase-based and Hierarchical Statistical Machine Translation [Wuebker & Huck$^+$ 12]
  - Hierarchical Phrase-Based Translation with Jane 2 [Huck & Peter$^+$ 12]
  - CharacTER: Translation Edit Rate on Character Level [Wang & Peter$^+$ 16]
  - Sisyphus, a Workflow Manager Designed for Machine Translation and Automatic Speech Recognition [Peter & Beck$^+$ 18]

- Other work
  - Sequence Labeling-based Reordering Model for Phrase-based SMT [Feng & Peter$^+$ 12]
  - Reordering: Advancements in Reordering Models for Statistical Machine Translation [Feng & Peter$^+$ 13]
  - (Hidden) Conditional Random Fields Using Intermediate Classes for Statistical Machine Translation [Lehnen & Peter$^+$ 13]

- International Evaluation Campaigns
  - The RWTH Aachen Machine Translation System for WMT 2013 [Peitz & Mansour$^+$ 13]
  - The RWTH Aachen Machine Translation Systems for IWSLT 2013 [Wuebker & Peitz$^+$ 13a]
  - The RWTH Aachen German-English Machine Translation System for WMT 2015 [Peter & Toutounchi$^+$ 15b]
  - The RWTH Aachen German to English MT System for IWSLT 2015 [Peter & Toutounchi$^+$ 15a]
  - The RWTH Aachen University English-Romanian Machine Translation System for WMT 2016 [Peter & Alkhouli$^+$ 16a]
  - The QT21/HimL Combined Machine Translation System [Peter & Alkhouli$^+$ 16b]
  - The RWTH Aachen Machine Translation System for IWSLT 2016 [Peter & Guta$^+$ 16]
  - The RWTH Aachen University English-German and German-English Machine Translation System for WMT 2017 [Peter & Guta$^+$ 17]
  - The QT21 Combined Machine Translation System for English to Latvian [Peter & Ney$^+$ 17]

2. Scientific Goals

The following scientific goals are pursued in this thesis:

- Small changes in the input of a phrase-based machine translation system can turn a previously seen phrase into an unseen phrase. Neural network models are, by design, able to handle these cases gracefully since they rely internally on continuous representations. Phrase-based systems, on the other hand, have to fall back to smaller phrases or even words. Our goal is to integrate neural models into the phrase-based system. This will make use of the greater flexibility provided by neural networks while keeping the benefits, like explainability, of the phrase-based approach. We also want to test methods to incorporate long-distance dependencies in feed-forward neural networks efficiently.

- The encoder-decoder approach with attention mechanism is limited to its previous internal state and the source encodings when computing the attention to output the next word. It does not have a direct feedback loop from the last used attention weights. In most languages, the last used source word can provide valuable information to decide which source word should be used next. We want to test if adding this information into the attention mechanism can improve the neural network performance. There are well-known methods used in statistical machine translation to create word alignments. We want to test if adding them as an additional target in the neural network model improves its performance.

  Neural network models do not require an explicit alignment since they are using the attention mechanism. There are applications like visualization or phrase-pair extraction that still require word alignments. One goal is to see if it is possible to modify the network in a way that allows us to create word alignments.

- Managing a large number of experiments and building the corresponding system in a reproducible way is a cumbersome and error-prone task, given their complicated setups. We want to develop a way to simplify the system setup management and allow a more effortless transfer of this knowledge to researchers working on a similar topic.

- To compare the effectiveness of our methods with state-of-the-art translation systems, we seek to evaluate them on publicly available large-scale tasks. This has been done within research projects and in public evaluations.

3. Preliminaries

The objective of this chapter is to review the preliminaries required to understand the methods used in this thesis. We start with an explanation of the terminology and notation. Next, we explain Bayes' decision rule and the log-linear model of the statistical machine translation approach. The basics of phrase-based machine translation are shown in the following section, and we conclude the chapter with an overview of the used evaluation metrics.

3.1 Terminology and Notation

To describe the methods presented in this work, we introduce the terminology and notations that we use. The language that we are translating from is often referred to as the source language or the source side. We use the notation f_1^J for a source sentence. f_j is referring to the word in the source sentence at position j, and J denotes the length of the source sentence. Similarly the language we are translating to is called the target language or target side with the notation of e_1^I for a target sentence, e_i for a single word, and I for the sentence length.

We need sentence-aligned bilingual data to build a translation system, i.e. data where we have for each sentence in the source language its translation in the target language. In addition, monolingual data refers to a set of sentences that are only available in a single language.

Sentence-aligned data can be annotated with word alignments. In that case, we have beside the parallel data, also the information which word on the source side is related to which word on the target side and the same for the reversed direction.

We use the notation a_1^J for alignments from the source side to the target side. a_j represents the index of the target word, which is aligned to the source word f_j. Similarly, b_1^I represents the alignment for all target words to the source side.

We refer to the approximation of the parameter used in a system or model as training. For a neural network, training means to approximate weights for the internal connections. If we want to create a phrase-based system, this means extracting the phrase table, and building the models which are part of the system (e.g. a language model). Decoding and search are both referring to the usage of a trained model to find the best translation of the given source sentence.

The provided parallel data are usually separated into training, development, and test data. The training data is the largest block and used to train the model. To estimate hyperparameters of the system, we are using the development data. While building a phrase-based system, it is used to optimize the log-linear feature weights. For neural network training, it is used to adjust the learning rate and select the best model. To evaluate the actual model performance, we use one or more test sets that are not in the model training at all.

We use the notation $Pr(\dots)$ for true unknown probability distributions and $p(\dots)$ for model distributions. We use the symbole '$\circ$' to represents the concatenation of two vectors.

3.2 Statistical Machine Translation

The overall goal of machine translation is to build a system that can automatically translate between natural languages, e.g., from Chinese to English. In the optimal case, a translation is generated by the computer without the need for human interaction. Given a sentence in the source language, it should return a sentence in the target language which preserves the meaning. Early approaches to reach this goal were made by using hand generated rules, which describe how to transform the given sentence into a sentence in the target language. Creating this rule set turned out to be increasingly hard with a growing number of rules and individual cases.

Cheap computational power and large amounts of texts translated into one or more languages let to the rise of data-driven translation. Data-driven translation attempts to create these translation rules from given training examples instead of generating and maintaining a vast set of rules by hand.

The first part of this thesis concentrates on Statistical Machine Translation (SMT), which is explained in this Section, while the second part concentrates on Neural Machine Translation, which is explained in Chapter 7. The basic concept behind SMT is to assign a probability to every possible translation and choose the most likely translation. NMT is using a neural network, and instead of a well-defined probability, the network score is used, which can be a probability. In many networks, this score resembles a probability, but it does not have to. The models are previously trained on the available example translations between the two languages, and depending on the model, additional data is used, e.g., text from the target language. SMT was first introduced by [Brown & Cocke$^+$ 88] and [Brown & Cocke$^+$ 90] with the IBM models described in Section 3.2.4.

3.2.1 Bayes' Decision Rule

The translation task can be defined as follows: Given a sentence in the source language $f_1^J = f_1 \dots f_J$ (a sequence of words f_j) find the sentence $\hat{e}_1^{\hat{I}} = e_1 \dots e_I$ in the target language that is most likely the correct translation. This is equivalent to finding the e_1^I that maximizes $Pr(e_1^I|f_1^J)$. Using Bayes' decision rule this can be written as follows:

$$\hat{e}_1^{\hat{I}} = \underset{e_1^I}{\operatorname{argmax}} \left\{ Pr(e_1^I|f_1^J) \right\} . \tag{3.1}$$

$\hat{e}_1^{\hat{I}}$ is the sentence with the highest probability, and therefore is selected as being the most accurate translation which also preserves the original meaning of f_1^J. The source side is usually referred to as f for French and the target side as e for English. This naming convention originates from the work done by [Brown & Cocke$^+$ 88], where a system to translate from French to English is trained. By applying Bayes' decision rule to Equation (3.1) we get the source-channel formulation introduced by [Brown & Della Pietra$^+$ 93].

$$\begin{aligned}
\hat{e}_1^{\hat{I}} &= \underset{e_1^I}{\operatorname{argmax}} \left\{ Pr(e_1^I|f_1^J) \right\} && (3.2)\\
&= \underset{e_1^I}{\operatorname{argmax}} \left\{ \frac{Pr(e_1^I) \cdot Pr(f_1^J|e_1^I)}{Pr(f_1^J)} \right\} && (3.3)\\
&= \underset{e_1^I}{\operatorname{argmax}} \left\{ Pr(e_1^I) \cdot Pr(f_1^J|e_1^I) \right\} && (3.4)
\end{aligned}$$

The denominator $Pr(f_1^J)$ in Equation (3.3) can be omitted since it is a constant w.r.t. e_1^I and does therefore not influence maximizing the argument. This leaves us with the language model $Pr(e_1^I)$

$$
\begin{aligned}
Pr(\text{Bills on ports were submitted}) \approx\ & p(\text{Bills}) \\
& \times p(\text{on}|\text{Bills}) \\
& \times p(\text{ports}|\text{Bills on}) \\
& \times p(\text{were}|\text{on ports}) \\
& \times p(\text{submitted}|\text{ports were})
\end{aligned}
$$

Figure 3.1: 3-gram language model example.

and the translation model $Pr(f_1^J|e_1^I)$ which can be trained independently. The translation model can be interpreted as a distance between the source and the target sentence, sometimes favoring ill-formed sentences. The language model can balance this effect by modeling the probability of e_1^I being a valid sentence in the target language.

3.2.2 Language Model

The language model $Pr(e_1^I)$ can be interpreted as a model of how likely the sentence e_1^I would occur in the target language. Using the chain rule we can rewrite this equation without loss of generality:

$$Pr(e_1^I) = \prod_{i=1}^{I} p(e_i|e_1^{i-1}) \tag{3.5}$$

Introducing the assumption that each word relies on its $n-1$ predecessors only, we can write the equation as:

$$\prod_{i=1}^{I} p(e_i|e_1^{i-1}) \approx \prod_{i=1}^{I} p_{\mathrm{n}}(e_i|e_{i-n+1}^{i-1}) \tag{3.6}$$

This assumption is clearly erroneous, e.g. the word *were* in Figure 3.1 needs to be changed to *was* in case *Bills* would be changed to *A bill*, but *were* does not even depend on *Bills* in a 3-gram language model. Even though this assumption is wrong, it is widely used to simplify the calculation of the language model.

A major problem for language models is the sparsity of the training data that is available, respectively that we are capable of processing. The sum of possible sentences exceeds by far the number of training samples seen by the model. As an example, for a 3-gram language model, given a vocabulary size of $W = 2 \cdot 10^4$ and a training corpus with $n = 10 \cdot 10^6$ running words, only $1.25 \cdot 10^{-4}\%$ of all 3-grams can be seen in training. Simply counting and normalizing the occurrences of a n-gram would therefore result in a probability of zero for all unseen n-grams. Even so they may still be possible in the target language. Smoothing and backing off are the two main approaches to address this issue.

Smoothing methods like Good-Turing discounting [Gale 95] or Kneser-Ney discounting [Kneser & Ney 95] take probability mass from n-grams with a high probability and shift these to unseen n-grams. These methods are employed during training to avoid n-grams with a probability of zero. Back-off models [Katz 87] are employed during scoring. For long n-grams is it likely that not the whole history was seen during training, however a shorter history may have occurred in the trainings data. The score is then computed by backing off to a shorter, less specific n-gram. A more thorough discussion about language modeling can be found in [Manning & Schütze 99].

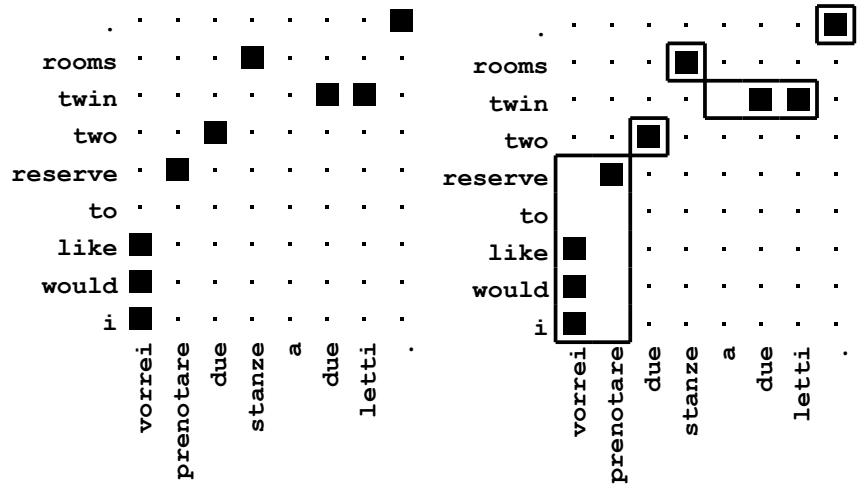


Figure 3.2: On the left side is an example of a word alignment and the right side show phrases which could be extracted from given the alignment.

3.2.3 Word Alignment

[Brown & Della Pietra$^+$ 93] introduced alignments as the hidden variable to connect corresponding words between source f_1^J and translation e_1^I. The usual interpretation is that if f_j is aligned to e_i, f_j has been translated to e_i. Even for humans, is it often difficult to give precise information which words correspond to each other. Special cases can be that a word is not aligned to any other word or to more than one.

We define an alignment as a binary matrix $\mathcal{A}$ of the size $I \times J$ for a given source sentence f_1^J and target sentence e_1^I:

$$\mathcal{A} = [a_{i,j}]$$
$$a_{i,j} = \begin{cases} 1 \text{ if } f_j \text{ is aligned to } e_i \\ 0 \text{ otherwise} \end{cases} .$$

An example for an alignment matrix is given in Figure 3.2. Note that alignments do not have to be monotonic, e.g. if e_i is aligned with f_j, than an alignment of e_n and f_m where $n < i \wedge m > j$ is still possible. Including the alignment in to the translation probability, we obtain:

$$Pr(f_1^J|e_1^I) = \sum_{\mathcal{A}} Pr(f_1^J, \mathcal{A}|e_1^I) \tag{3.7}$$
$$\approx \max_{\mathcal{A}} Pr(f_1^J, \mathcal{A}|e_1^I) \tag{3.8}$$

A maximum approximation as in Equation (3.8) is often used due to the huge number of possible alignments. If each word in e_1^I is only aligned to one word in f_1^J, then the alignment can also be written as vector $a_1^J := a_1, \ldots, a_j, \ldots, a_J$. The target word e_{a_j} here, is aligned to the source word f_i. This notation is useful for the single-word-based models as presented in Section 3.2.4.

3.2.4 Single-Word-Based Model

The translation model $Pr(f_1^J|e_1^I)$ is used to model the probability of f_1^J being a translation of e_1^I. Single-word-based models use the previously introduced alignment as a hidden variable for a convenient factorization of the lexical translation probability:

$$Pr(f_1^J|e_1^I) = \sum_{a_1^J} Pr(f_1^J|a_1^J, e_1^I, J) \tag{3.9}$$

$$= \sum_{a_1^J} \prod_{j=1}^{J} Pr(f_j|f_1^{j-1}, a_1^J, e_1^I, J) \tag{3.10}$$

$$= \max_{a_1^J} \prod_{j=1}^{J} Pr(f_j|f_1^{j-1}, a_1^J, e_1^I, J) \tag{3.11}$$

$$\approx \max_{a_1^J} \prod_{j=1}^{J} Pr(f_j|e_{a_j}) \tag{3.12}$$

The distribution is reduced to the probability of a source word given the aligned target word as in Equation (3.12), in single-word-based modeling. The approximation uses the assumption that every word in the target sentence only relies exactly on one word in the source sentence, and therefore no context information is taken into account.

Different single-word-based translation models that mainly differ in the alignment model have been proposed. Starting from a uniform distribution in the IBM Model 1 [Brown & Della Pietra$^+$ 93]:

$$Pr(a_1^J|e_1^I, J) = \prod_{j=1}^{J} Pr(a_j|a_1^{j-1}, e_1^I, J) \tag{3.13}$$

$$Pr(a_j|a_1^{j-1}, e_1^I, J) \approx p(a_j|I) = \frac{1}{I+1} \tag{3.14}$$

There are more sophisticated models like the Hidden Markov Model (HMM) [Vogel & Ney$^+$ 96] which incorporates the previous alignment:

$$Pr(a_j|a_1^{j-1}, e_1^I, J) \approx Pr(a_j|a_{j-1}, I, J) \tag{3.15}$$

and models that include more context in the alignment like the IBM Model 4 and 5. A comparison of different single-word-based translation models can be found in [Och & Ney 03].

3.2.5 Log-Linear Modeling

The log-linear model [Och & Ney 02] gives a more flexible decomposition of Bayes' decision rule. It allows for an arbitrary number of features in the form of $h_m(f_1^J, e_1^I)$. These features could be the previously described language model, translation model, or other models which provide useful scores, as illustrated in Figure 3.3. Each feature has a corresponding scaling factor λ_m.

$$Pr(e_1^I|f_1^J) = \frac{\exp\left(\sum_{m=1}^{M} \lambda_m h_m(f_1^J, e_1^I)\right)}{\sum_{\tilde{e}_1^I} \exp\left(\sum_{m=1}^{M} \lambda_m h_m(f_1^J, \tilde{e}_1^I)\right)} \tag{3.16}$$

Using the log-linear model we can now rewrite Bayes' decision rule:

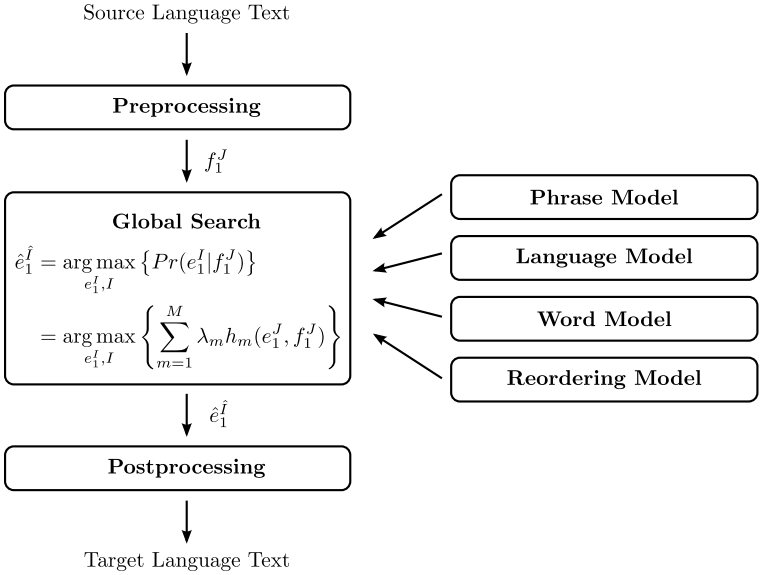


Figure 3.3: A graphic illustration of how a translation system is built.

$$\hat{e}_1^{\hat{I}} = \underset{e_1^I}{\text{argmax}} \left\{ Pr(e_1^I|f_1^J) \right\} \tag{3.17}$$

$$= \underset{e_1^I}{\text{argmax}} \left\{ \frac{\exp\left(\sum_{m=1}^{M} \lambda_m h_m(f_1^J, e_1^I)\right)}{\sum_{\tilde{e}_1^I} \exp\left(\sum_{m=1}^{M} \lambda_m h_m(f_1^J, \tilde{e}_1^I)\right)} \right\} \tag{3.18}$$

$$= \underset{e_1^I}{\text{argmax}} \left\{ \exp\left(\sum_{m=1}^{M} \lambda_m h_m(f_1^J, e_1^J) \right) \right\} \tag{3.19}$$

$$= \underset{e_1^I}{\text{argmax}} \left\{ \sum_{m=1}^{M} \lambda_m h_m(f_1^J, e_1^J) \right\} \tag{3.20}$$

$$\tag{3.21}$$

We can again omit the denominator and also the exponential function. The λ_m can be optimized directly towards a desired metric as the Bleu score, an evaluation metric described in Section 3.3.2, on a small development corpus. The ability to add new features and to train the according λ_m is the key advantage over the original source-channel approach.

Optimization of λ_m on an error criterion like the Bleu score is called Minimum Error Rate Training (MERT). Using MERT gives state-of-the-art results in machine translation where improvements in the translation quality are due to the models used as features in the log-linear model.

3.2.6 Phrase-Based Translation

Single-word-based translation using a IBM or HMM model as described in Section 3.2.4 has a major limitation. Starting at an alignment a_1^J as used by some models, the lexical probability is modeled using $p(f_j|e_{a_j})$. The probability of f_j depends, therefore only on the aligned word e_{a_j}, no other context information is taken into account. Phrase-based translation systems [Och & Tillmann$^+$ 99, Zens & Och$^+$ 02, Koehn & Och$^+$ 03] were introduced to reduce this problem.

What sets the phrase-based translation models apart from the IBM and HMM models is that phrase-based will model sequences of words, called phrases. Note that in the statistical machine translation context, phrases are simply groups of consecutive words on the source and target sides and are not based on a linguistic notion.

We need a bilingual corpus with a word alignment as described in Section 3.2.3 for the phrase extraction. This alignment can be obtained by training the IBM models. We can use this alignment to extract phrase pairs $\langle \hat{f}, \hat{e} \rangle$, where $\hat{f}$ and $\hat{e}$ are a consecutive group of words on the source and target sides. All words in $\hat{f}$ must only be aligned to words in $\hat{e}$ and vice-versa.

More formally, we can write the set of phrases that can be extracted from a given sentence pair (f_1^J, e_1^I) and a corresponding word alignment $\mathcal{A}$ (Section 3.2.3) as follows:

$$\mathcal{P}(f_1^J, e_1^I, \mathcal{A}) = \big\{ (f_{j_1}^{j_2}, e_{i_1}^{i_2}) \mid j_1, j_2, i_1, i_2 \text{ s.t.} \tag{3.22}$$

$$\forall (j,i) \in \mathcal{A} : j_1 \leq j \leq j_2 \Leftrightarrow i_1 \leq i \leq i_2 \tag{3.23}$$

$$\wedge \exists (j,i) \in \mathcal{A} : (j_1 \leq j \leq j_2 \wedge i_1 \leq i \leq i_2) \big\} . \tag{3.24}$$

Figure 3.4 shows a possible word alignment between an Italian and an English sentence with the phrases that are to be extracted from it. Extracting all phrases that fulfill the definition in Equation (3.24) from each sentence pair in the training data provides us with a large number of phrases. A more detailed description of phrase extraction and heuristics can be found in [Koehn & Och$^+$ 03]. The set of extracted phrases from a corpus is the union of the set of phrases extracted from each of its parallel sentences. A probability for each phrase pair is then estimated, using relative frequencies.

During the translation process the source sentence will be segmented into single phrases. Using the phrase table each phrase will be translated. The phrases are then permuted [Och & Ney 04] and composed again to one sentence. The segmentation of a sentence pair $(f_1^J | e_1^I)$ into K phrase pairs is defined as:

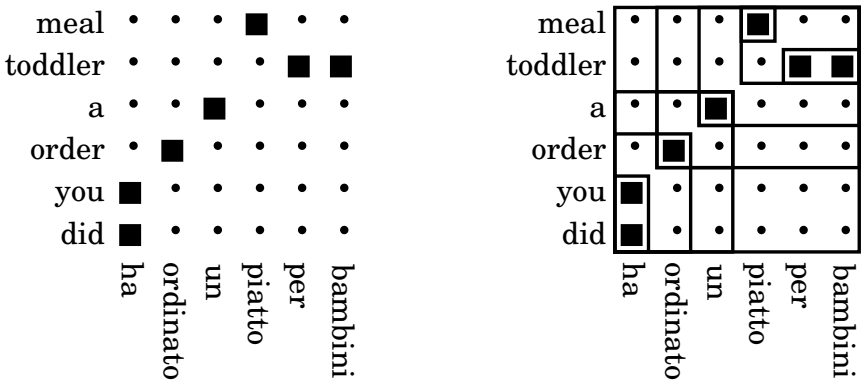
$$k \rightarrow s_k := (i_k; b_k, j_k), \text{ for } k = 1 \ldots K$$

Where i_k denotes the end position of the target phrase with index k and the pair (b_k, j_k) denotes the start and end position of the source phrase, which is aligned to the target phrase k. The segmentation is restricted to cover every word on the target and source side exactly once. There are no gaps or overlaps allowed. Formally:

$$\bigcup_{k=1}^{K} \{b_k, \ldots, j_k\} = \{1, \ldots, J\}$$

$$\{b_k, \ldots, j_k\} \cap \{b_{k'}, \ldots, j_{k'}\} = \emptyset \; \forall k \neq k'$$

The segmentation s_1^K is introduced as a hidden variable in the translation model, remark that the segmentation also contains the information about phrase reordering.



Alignment Standard phrases

Figure 3.4: Possible standard phrases (right), given a word alignment (left)

$$
\begin{aligned}
Pr(e_1^I|f_1^J) &= \sum_{s_1^K} Pr(e_1^I, s_1^K|f_1^J) \\
&= \sum_{s_1^K} \frac{exp\left(\sum_{m=1}^M \lambda_m h_m(e_1^I, s_1^K; f_1^J)\right)}{\sum_{e_1'^{I'}, s_1'^{K'}} exp\left(\sum_{m=1}^M \lambda_m h_m(e_1'^{I'}, s_1'^{K'}; f_1^J)\right)} \\
&\approx \max_{s_1^K} \frac{exp\left(\sum_{m=1}^M \lambda_m h_m(e_1^I, s_1^K; f_1^J)\right)}{\sum_{e_1'^{I'}, s_1'^{K'}} exp\left(\sum_{m=1}^M \lambda_m h_m(e_1'^{I'}, s_1'^{K'}; f_1^J)\right)}
\end{aligned}
$$

A maximum approximation is used again, as with the alignment model, to eliminate the need to sum over the possibly large number of segmentations. We can drop the denominator since this term is independent of the target sentence e_1^I.

$$
\hat{e}_1^{\hat{I}} = \underset{I, e_1^I}{\operatorname{argmax}} \left\{ \max_{s_1^K} \sum_{m=1}^M \lambda_m h_m(e_1^I, s_1^K; f_1^J) \right\}
$$

3.3 Evaluation Metrics

The automatic evaluation of translations is an essential topic in machine translation. Human evaluation is time-consuming, expensive, and not always possible. To evaluate vast amounts of translations, as created by translation systems automatic, fast, and language-independent metrics were developed. Besides evaluations of the final translation result are these metrics used during the optimization process of the system. The challenge in developing a metric to evaluate the performance of a translation system is that there can be many correct translations for a single sentence, while at the same time, often just one reference translation is given. Even if multiple correct translations are provided, not all correct translations will be covered. This reason makes the *Word Error Rate* [Hunt 90], a metric commonly used in automatic speech recognition, barely used in current machine translation research.

Many error metrics for machine translation have been proposed. Some sophisticated metrics proposed for statistical machine translation are:

- NIST [Doddington 02]
- ROUGE [Lin 04]
- METEOR [Banerjee & Lavie 05]
- Beer [Stanojevic & Sima'an 14]
- CharacTER [Wang & Peter$^+$ 16]

We base our results in this thesis on Bleu and TER, which are described in the following subsections.

3.3.1 Translation Edit Rate

The translation edit rate (TER)[Snover & Dorr$^+$ 05, Snover & Dorr$^+$ 06] is defined as the number of edit operations to convert the hypotheses into the reference divided by the number of words in the reference.

$$TER = \frac{\#\text{ of edit operations}}{\#\text{ of reference words}}$$

Possible operations for TER are similar to the Levenshtein distance: the insertion, the deletion, or the substitution of words. In addition, the shifts of word sequences are allowed. A shift moves a contiguous sequence in the hypotheses to a different place. If more than one reference is provided, the one with the smallest TER score is chosen. TER has a tendency to favor short translations over longer translations. This happens since the likelihood of adding a wrong word, therefore increasing the number of edit operations, grows with the length of the translation.

3.3.2 Bilingual Evaluation Understudy

The bilingual evaluation understudy, generally referred to as Bleu, was proposed by [Papineni & Roukos$^+$ 02] and is probably the most common way to evaluate the quality of automatic translations. Bleu tries to model the translation quality by calculating the mean over a modified n-gram precision combined with a brevity penalty. Formally:

$$\textsc{Bleu}(e_1^I, \hat{e}_1^{\hat{I}}) = BP(I, \hat{I}) \cdot \exp\left(\frac{1}{N}\sum_{n=1}^{N} p_n(e_1^I, \hat{e}_1^{\hat{I}})\right) \tag{3.25}$$

with

$$BP(I, \hat{I}) = \begin{cases} 1 & \text{if } I \geq \hat{I} \\ \exp(1 - \hat{I}/I) & \text{if } I < \hat{I} \end{cases} \tag{3.26}$$

$$p_n(e_1^I, \hat{e}_1^{\hat{I}}) = \frac{\sum_{w_1^n} \min\left\{C(w_1^n|e_1^I), C(w_1^n|\hat{e}_1^{\hat{I}})\right\}}{\sum_{w_1^n} C(w_1^n|e_1^I)} \tag{3.27}$$

$$C(w_1^n|e_1^I) = \#\text{ of } w_1^n \text{ in } e_1^I \tag{3.28}$$

For a given hypothesis e_1^I and a reference translation $\hat{e}_1^{\hat{I}}$, the n-gram precision $p_n(e_1^I, \hat{e}_1^{\hat{I}})$ is calculated by counting all n-grams occurring in the hypothesis and reference translation divided by the number of n-grams in the hypothesis shown in Equations (3.27) and (3.28). The geometric mean overall n-grams with $n \leq N$ is then multiplied with the brevity penalty, Equation (3.25). The brevity penalty is used to counteract the effect that precision alone would favor short translation,

Equation 3.28. The BLEU score is computed on the whole corpus different from the Levenshtein distance, which is evaluated on a sentence level.

BLEU showed a high correlation with human evaluation [Papineni & Roukos$^+$ 02] when proposed and is still a widely used evaluation metric, which can be seen in most publications about machine translation.

3.3.3 Alignment Error Rate

The Alignment Error Rate (AER) is a metric derived from the F-Measure, which is used to evaluate the alignment quality [Och & Ney 03]. The human alignment can provide two different values for each alignment point. S for "sure" alignments and P for "possible" alignment. Given these human alignments, the AER is defined as:

$$AER(S, P; A) = 1 - \frac{|A \cap S| + |A \cap P|}{|A| + |S|}$$

3.3.4 Soft Alignment Error Rate

The Soft Alignment Error Rate (SAER) is a modification of the AER by [Tu & Lu$^+$ 16] made to handle alignments that are given as a probability. This is the case if the attention is interpreting as alignment.

$$SAER(M_S, M_P; M_A) = 1 - \frac{|M_A \odot M_S| + |M_A \odot M_P|}{|M_A| + |M_S|}$$

where $\odot$ is the element wise matrix multiplication, and $|M|$ is the sum of all elements in M. The reference matrix M_S and M_P are generated using the human alignment. They are filled with zeros and have the same shape as M_A. $M_P(i, j)$ is set to 1 if $(i, j) \in P$ and similar for M_S and S. Additionally $M_P(i, j)$ is set to 0.5 if $(i, j) \in P/S$.

4. Neural Networks

Artificial Neural Networks, in this work, also referred to as just Neural Networks, are computational models that were inspired by mechanisms found in the human brain and are mostly used to model classification tasks. Artificial neural networks have first been introduced in 1943 [McCulloch & Pitts 43]. While they could only be used for toy tasks during the first years, they became widely used in the last two decades due to their high flexiblty and the availability of fast and cheaply computation resources. They consist of many perceptrons that are connected to another. If enough perceptrons connected to a given perceptron fire, this perceptron should also fire. A perceptron is the smallest unit in a neural network. Neurons, as they occur in the nervous system, inspire their design. While the simple design is less potent than a natural neuron, one natural neuron can be enough to simulate an XOR gate [Gidon & Zolnik$^+$ 20], it is powerful enough for many applications and can be efficiently computed.

4.1 Multilayer Perceptron

The multilayer perceptron is a basic neural network architecture [Rumelhart & Hinton$^+$ 86, Werbos 88, Bishop 95]. The perceptrons are here structured into multiple layers and connected. If they are connected without circles, which allows the information only to move forward, they are referred to as feed-forward neural networks.

Connecting two layers is equal to connecting the output of each node in the first layer with the input of each node in the second layer, if we are working with a fully connected network. Fully connected networks are the widest spread networks. Full connections simplify the computation since they can be efficiently modeled as matrix multiplication.

At least one layer is used as an input layer. The activation of an input layer is given by an external vector as input to the computation. Similarly, the model contains at least one output layer. The values of the output layer are the prediction of the model. During training, the prediction is compared to the known output, and the model weights are updated to reduce the error. Their ability to create a continuous representation of the inputs allows them to generalize well to unseen events.

4.1.1 Input Layer

One challenge of using neural networks for text is how the character strings can be turned into vectors that are used as inputs and outputs of the neural network. This is usually solved by using one-hot vectors [Bengio & Ducharme$^+$ 03].

The inputs and outputs of neural networks are defined as the activation of the corresponding nodes. Since we usually group nodes layer-wise, we can see the activations of the input layer as an input vector and the activation of the output layer as an output vector. In the first layer of the neural network, these one-hot vectors are projected into a smaller vector space. This layer is

referred to as the projection layer. The input vectors have the size of the input vocabulary $|V_e|$ and contain only zeros except for one position, the hot position, which is set to 1:

$$\hat{e}|_m = \begin{cases} 1, & \text{if } m = k \\ 0, & \text{otherwise} \end{cases}$$

To convert the one-hot vector into a more compact and continuous representation it is multiplied with the embedding matrix $A_1 \in \mathbb{R}^{d_e \times |V_e|}$, where d_e is the dimensionality of the embedding vector. The multiplication of the embedding matrix with a one-hot vector effectively selects one row from the embedding matrix.

$$E_e : \mathbb{R}^{|V_e|} \to \mathbb{R}^{d_e}$$
$$E_e(\hat{e}) = A_1 \hat{e}$$

In practice this operation can be performed much faster than a full matrix multiplication. Multiple one-hot vectors can be feed to the network by concatenating their outputs, to provide multiple input vectors to the neural network:

$$y^{(1)} = E_e(\hat{e_1}) \circ E_e(\hat{e_2}) \circ \ldots \circ E_e(\hat{e_n})$$

where $\circ$ represents the concatenation operation and $y^{(1)}$ the first hidden layer. From now on, we will refer to the output of the l-layer using the superscript notation: $y^{(l)}$.

4.1.2 Short List

To create a one-hot vector for the whole vocabulary would result in a massive input and output matrix. The computational cost to generate the input y^1 would be constant since it is just a table lookup. The computation cost, on the other hand, grows linearly with the output. An additional problem is that this would result in many words that are barely seen during training. On the input side, these words would have barely trained word embeddings, while on the output side, these words would have a close to zero chance of being produced. One way to reduce the problem is to build a shortlist and train only on the most frequent words and map all other words to a predefined unknown token. While this limits the expressiveness of the network, it makes the whole network easier to train and to apply. It is also possible to cluster all other words into classes to either predict only the class probability or have an additional step to separate the word probability inside each class [Morin & Bengio 05, Son & Allauzen$^+$ 12]. A common method is also to split words into smaller pieces and ensure that each of these pieces has been seen frequently enough [Sennrich & Haddow$^+$ 15, Kudo & Richardson 18].

4.1.3 Hidden Layer

Layers that have no direct connection to the input or output are called hidden layers. The hidden layer l takes the outputs from the previous layer $y^{(l-1)}$, multiplies them with its weight matrix A_l, adds the local bias b_l, and applies an activation function ϕ^l element-wise:

$$y^{(l)} = \phi(A_l y^{(l-1)} + b_l)$$

Common activation functions are the hyperbolic tangent, sigmoid, and the linear rectifier. An activation function should be continuous, nonconstant, differentiable, and monotonically-increasing. The derivative of the activation function is important for the backpropagation algorithm (Section 4.3.3).

Hyperbolic tangent approaches 1 for $x \to \infty$ and -1 for $x \to -\infty$. It is defined as:

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

and its derivative is given by:

$$\tanh'(x) = 1 - \tanh^2(x)$$

Sigmoid approaches 1 for $x \to \infty$ and 0 for $x \to -\infty$. It is defined as:

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

its derivative is:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

The rectifier is defined as:

$$\text{rect}(x) = max(0, x)$$

it derivative is technically undefined at 0, but in practical applications, it is set to 0:

$$\text{rect}'(x) = \begin{cases} 1, & \text{if } x < 0 \\ 0, & \text{otherwise} \end{cases}$$

4.1.4 Output Layer

The output layer of a network is used to predict the probability of the next word. This is done similar to the input of a neural network by mapping the possible vocabulary to a vector. Since this output vector has the size of the active vocabulary $|V|^e$, it becomes, together with the input layer, easily the largest weight matrix in a regular feed-forward network.

While the input layer can be computed very efficiently, most methods require to compute the full, computationally expensive, output layer each time the network applied.

4.1.5 Softmax

To ensure that the values of the output layer can be interpreted as probability distribution, we need to ensure they fulfill the definition of a probability distribution:

$$p_\theta(c|x) > 0 \text{ and } \sum_{c \in C} p_\theta(c|x) = 1 \tag{4.1}$$

This requires a normalization step, which is usually done using a softmax function after the output layer:

$$p(c|x) = \frac{e^{y^l(x)_c}}{\sum_k e^{y^l(x)_k}}$$

This guarantees the properties needed for a probability distribution.

The cost of a softmax layer can be reduced by splitting them into multiple smaller layers by dividing all words into classes. This allows computing two smaller hidden layers, first the class of a word and then the final probability of the word given the class, instead of one large layer [Mikolov & Kombrink$^+$ 11].

An alternative to these methods is noise contrastive estimation [Gutmann & Hyvärinen 10, Mnih & Teh 12, Vaswani & Zhao$^+$ 13] which avoids computing the whole output layer.

4.1.6 Deep Networks

If multiple hidden layers inside of an MLP are stacked on top of each other, the resulting network is commonly referred to as a deep neural network. The Universal Approximation Theorem proven by Cybenko in 1989 [Cybenko 89] can be used to demonstrate that one hidden layer is enough to approximate any continuous function as long as this hidden layer is large enough. In practice, neural networks with multiple hidden layers regularly outperform shallow neural networks.

Training very deep networks can be challenging, but methods like layer-wise pretraining [Bengio & Lamblin$^+$ 07, Seide & Li$^+$ 11] or the usage of *restricted Boltzmann machines (RBM)*[Hinton & Osindero$^+$ 06] have shown that these problems can be overcome. With some changes to its structure, even very deep networks with hundred layers can be successfully trained [Srivastava & Greff$^+$ 15].

4.2 Recurrent Neural Networks

Recurrent neural networks are an extension of feed-forward networks. While feed-forward networks can be potent tools, they do have limitations. Their fixed input and output requires that the task needs to be converted to fit this structure.

Recurrent neural networks allow for inputs and outputs of arbitrary length by using the hidden state of the previous time step as an additional input of a recurrent layer.

4.2.1 Recurrent Layer

Recurrent layers add an additional dimension to the notation, commonly referred to as time. The output of layer l in time step i is represented as $y_i^{(l)}$, and the output of the same layer in the previous time step is $y_{i-1}^{(l)}$. The recurrent layer is now defined as:

$$y_i^{(l)} = \phi(A_l(y_i^{(l-1)} \circ y_{i-1}^{(l)}) + b_l)$$

where A_l is its weight matrix, b_l the bias vector, and ϕ, the element-wise activation function similar to the hidden layer.

4.2.2 Long Short-Term Memory (LSTM) Layer

The direct implementation of recurrent layers sufferers from vanishing and exploding gradients [Bengio & Simard$^+$ 94, Hochreiter & Bengio$^+$ 01, Pascanu & Mikolov$^+$ 13]. The harm of exploding gradients can be significantly reduced by gradient clipping. Avoiding the vanishing gradient is more complicated. The *long short-term memory* (LSTM) layer [Hochreiter & Schmidhuber 97], is a common way to avoid these problems by protecting the internal state with a gating mechanism.

The *gated recurrent unit* (GRU) is a variant introduced by [Cho & van Merrienboer$^+$ 14b], which follows a similar approach. The LSTM layer performed empirically either better [Irie & Tüske$^+$ 16] or equally well [Chung & Gulcehre$^+$ 14]. We are using LSTM layers for all recurrent neural networks in this thesis due to this.

4.3 Neural Network Training

The Neural network models used in this thesis are trained using supervised learning. Supervised learning requires a large number of known network inputs with the correct output. This is in contrast to unsupervised learning, which does not require the correct output to learn.

For supervised learning, the network predicts an output given one of the inputs of the training samples. This predicted output is in the next step compared to the correct output. Out of the difference, a loss is computed using a loss function. Each model parameter is then updated to reduce this loss.

4.3.1 Loss Function

The most common loss functions are the mean squared error (MSE) and the cross-entropy criterion. The mean squared error is more general than cross-entropy since it is merely minimizing the distance between two similar dimensioned vectors without any further limitations.

Since we are only handling cases with class output in this thesis, we will also limit the example to classes. Given our annotated training data is given as an input vector x_n and the correct output class is c_n for N training samples. The mean squared error is then defined as follows:

$$\mathcal{L}_{\text{MSE}} := \frac{1}{N}\sum_{n=1}^{N}\sum_{c}[p_\theta(x_n, c) - \delta(c, c_n)]^2 \tag{4.2}$$

When predicting classes, the neural network is used to model a probability distribution.

A softmax output layer is used (Section 4.1.5), to ensure the network output fulfills all requirements for a probability distribution. This means that the neural network model output, $p_\theta(c|x)$, is constrained to:

$$p_\theta(c|x) > 0 \text{ and } \sum_{c\in C} p_\theta(c|x) = 1 \tag{4.3}$$

in this case, we can also make use of the cross-entropy criterion:

$$\mathcal{L}_{CE} := -\frac{1}{N}\sum_{n=1}^{N} log p_\theta(c_n|x_n) \tag{4.4}$$

The combination of the softmax output layer and cross-entropy criterion has numerical advantages compared to the mean squared error and is, therefore, in most cases, preferred. All neural networks in this work are training using a softmax output layer and cross-entropy output unless specified otherwise.

4.3.2 Gradient Descent

The weight update of the network is done by computing the gradient for each parameter with respect to the loss function. It is always possible to compute the gradient of a neural network since it is a composition of differentiable functions. The gradient always points to the steepest ascent in this multi-dimensional vector space. Therefore updating the parameters with the negative error gradient will lead to a minimum in this space. There is, however, no guarantee to reach a global minimum.

Computing the gradient for the whole training data and update the parameters once is called *batch gradient descent*. Updating the parameters for each training sample is called *stochastic gradient descent*. Batch gradient descent takes to long for one parameter update if a nontrivial number of training samples is used, which lead to a very slow convergence. Stochastic gradient descent, on the other hand, tends to be less stable. In practice, the best results are reached by using a compromise: Batches that only cover a small part of the whole training data, the so-called

mini-batch gradient descent. The work by [Wilson & Martinez 03] demonstrated on 27 learning tasks that batch training generates similarly performing neural network models, while being in general faster to compute and using less memory. In this thesis, all models are trained using *mini-batch gradient descent.*

4.3.3 Backpropagation

The backpropagation algorithm [Rumelhart & Hinton$^+$ 86, Werbos 88, Williams & Zipser 95] presents a way how the gradient for each parameter of the neural network can be computed. Using the chain rule for partial derivates, it propagates the loss from the last layer one layer at the time back until it reaches the input layer. This method can be applied as long as each layer and activation function is differentiable.

4.3.4 Backpropagation Through Time

Updating the weights for recurrent neural networks presents some particular challenges. Some have been address with *real time recurrent learning* [Robinson & Fallside 87] and *backpropagation through time* [Williams & Zipser 95, Werbos 90]. Backpropagation through time back propagates the gradient not only to the previous layer but also to the previous time step. This is done recursively until either the first time step has been reached or a previously set limit of time steps has been reached.

LSTM and GRU layers are both fully differentiable. Therefore backpropagation through time can be used without modification with them.

4.3.5 AdaDelta

Choosing a learning rate can have a significant impact on the neural network training. A too-small learning rate can cause the training to converge very slowly or even get stuck in a local minimum. On the other hand, networks with a too large learning rate often either do not converge or even fail by causing numerical problems.

Algorithms like *Adagrad* [Duchi & Hazan$^+$ 11] offer a solution to this problem by performing smaller updates on frequently changed parameters and larger updates on infrequently changed parameters. This makes the previously frequent changed parameters more stable while at the same time allowing the infrequently changed parameters still to be updated.

A modified version of Adagrad is *AdaDelta*[Zeiler 12]. It tries to prevent a monotonic shrinking of the learning rate during training, which happens with Adagrad. The exact algorithm used by AdaDelta for an update at time step t is:

$$
\begin{aligned}
g_t &= \nabla \mathcal{L}_{\theta_t} \\
n_t &= \beta n_{t-1} + (1-\beta) g_t^2 \\
r(n_t) &= \sqrt{n_t + \epsilon} \\
\Delta\theta_t &= \frac{-\eta}{r(n_t)} g_t \\
s_t &= \beta s_{t-1} + (1-\beta)\Delta\theta_t^2 \\
r(s_{t-1}) &= \sqrt{s_{t-1} + \epsilon} \\
\theta_{t+1} &= \theta_t - \frac{r(s_{t-1})}{r(n_t)} g_t
\end{aligned}
$$

where the current model parameters are stored in θ, $\nabla \mathcal{L}_{\theta_t}$ is the gradient for the current loss, η is the learning rate, and β a hyperparameter.

4.3.6 Adaptive Moment Estimation (Adam)

Adaptive Moment Estimation [Kingma & Ba 14] follows an idea similar to AdaDelta but also integrates an exponentially decaying average of past gradients. This results in similar behavior as momentum. In experiments this helped networks to converge faster and to a better optimum. This and a comparison of multiple optimization algorithms applied to neural machine translation can be found in [Bahar & Alkhouli$^+$ 17].

The exact algorithm used by Adam for an update at time step t is:

$$
\begin{aligned}
g_t &= \nabla \mathcal{L}_{\theta_t} \\
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
n_t &= \beta_2 n_{t-1} + (1-\beta_2) g_t^2 \\
\hat{n}_t &= \frac{n_t}{1-\beta_2^t} \\
\theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{n}_t} + \epsilon} \hat{m}_t
\end{aligned}
$$

where the current model parameters are stored in θ, $\nabla \mathcal{L}_{\theta_t}$ is the gradient for the current loss, η is the learning rate, and β_1 and β_2 are hyperparameters.

4.3.7 Regularization

Regularization methods can be applied to avoid overfitting to the training data. A simple approach is to restrict the weights from growing too large by adding the L^1 or L^2 norm of the weight matrices to the objective function [Krogh & Hertz 91].

A very prominent method which is also used in this thesis is *dropout* [Hinton & Srivastava$^+$ 12, Srivastava & Hinton$^+$ 14]. Dropout sets the output of some nodes during training randomly to zero, basically removing them from the neural network. This forces the neural network to not rely heavily on single nodes, but to spread the information across multiple nodes. This method has shown to improve the neural network performance and make the resulting neural networks more robust.

5. Feed-Forward Models in Phrase-Based Systems

In this chapter we describe multiple variants of feed-forward neural networks (Section 4.1) and their combination with phrase-based translation (Section 3.2.6). After an introduction given in Section 5.1 and an overview of related work (Section 5.2), we describe different models (Section 5.3), and how to integrate a longer context using bag-of-words features (Section 5.4). This chapter concludes with a description of the contribution of the author of this thesis (Section 5.5).

5.1 Introduction

Integrating neural networks with phrase-based machine translation raises a few challenges. Mapping words to neural network input and output vectors can be solved using one-hot-encoding [Bengio & Ducharme$^+$ 03]. This also reduces the first matrix multiplication of the input vector to a simple table lookup. To use the neural network output as a probability, we need to normalize it. This can be done using a softmax output, as described in Section 4.1.5.

Another problem is how to combine the models. A language model can be used as a regular count-based n-gram language model. For translation modeling, we need to know which word we are translating. This can be done using the word alignment of the phrase-based system similar to [Devlin & Zbib$^+$ 14], or by creating a separate alignment.

On the implementation side, the question arises if the model should be included in rescoring or during decoding. Rescoring allows for more complex models since the whole target sentence is known, and the model will only be used to score the translation hypothesis that made it into the n-best list. The implementation also tends to be more straightforward since it does not require to modify the decoder. Implementing a model in decoding has the advantage that it can utilize the whole search space and is not limited to sentences that made it into the n-best list, which often allows for better results.

5.2 Related Work

Neural networks for language models were proposed in 1989 by [Nakamura & Shikano 89] and carried forward by [Bengio & Ducharme$^+$ 03]. Recurrent networks were used to show that inputs of arbitrary length can be handled successfully with neural networks in machine translation [Mikolov & Karafiát$^+$ 10]. Translation models and language models were effectively combined in [Auli & Galley$^+$ 13]. The n-of-N encoding input vectors used to represent the bag-of-words features are described in [Irie & Schlüter$^+$ 15]. Several groups have reported significant improvements over state-of-the-art baselines when combining phrase-based translation with feed-forward neural network models [Schwenk & Déchelotte$^+$ 06, Schwenk 12]. The integration into decoding was presented by [Vaswani & Zhao$^+$ 13, Devlin & Zbib$^+$ 14]. The performance of recurrent neural

network models in combination with phrase-based machine translation has been demonstrated in [Sundermeyer & Alkhouli$^+$ 14].

The main drawback of a feed-forward neural network model compared to a recurrent neural network model is that it can only have a limited context length on source and target sides. Using the *Bag-of-Words* (BoW) as an additional input of neural network-based language models offers a way to circumvent this problem. [Mikolov & Joulin$^+$ 15] have achieved very similar perplexities on automatic speech recognition tasks in comparison to the long short-term memory (LSTM) neural network, whose structure is much more complex, by using a BoW as input.

5.3 Models

We show three different basic models that we combine with the phrase-based system. They are defined by their different inputs.

- The *Language Model* which works only on the target side (Section 5.3.1).
- The *Translation Model* which takes its input from the source side and makes its predictions on the target side (Section 5.3.2).
- The *Joint Model* which takes its input from the source and target side and makes its predictions on the target side (Section 5.3.3).

5.3.1 Language Models

Neural networks as language models can be modeled similar to count-based language models using n-grams. The advantage of using a neural network for this task is that methods like leaving one out [Kneser & Ney 95] to handle unseen events are not needed. In neural networks, the last $n-1$ words are used as input of the neural network to predict the next word. In the implementation used in this work, all $n-1$ inputs share one embedding matrix, and the output is merged in the next step. After applying one or more hidden layers, the last layer is a softmax layer to predict the next word. Figure 5.1 shows how a trigram language model using feed-forward neural networks is constructed. The approximation to ignore all words more than n words before the current word, as it is done with an n-gram language model, can be formalized like this:

$$\begin{aligned} Pr(e_1^I) &= \prod_{i=1}^{I} Pr(e_i|e_1^{i-1}) \\ &\approx \prod_{i=1}^{I} Pr(e_i|e_{i-n-1}^{i+1}) \end{aligned}$$

where $Pr(e_1^I)$ is the probability of the whole sequence and $Pr(e_i|e_1^{i-1})$ is the probability of the word e_i given all words before this position.

Neural networks can also be used to include an arbitrary long history if they are modeled using recurrent neural networks. In this case, the last word is used as input to the neural network, which should predict the next word. All previous words are by design stored in the hidden state of the recurrent neural network. The networks are, in principle, able to incorporate more information than feed-forward models which have a limited history. On the other hand, they are slower and harder to train, which is why the experiments in this work, which are performed in combination with phrase-based models, are produced with feed-forward models.

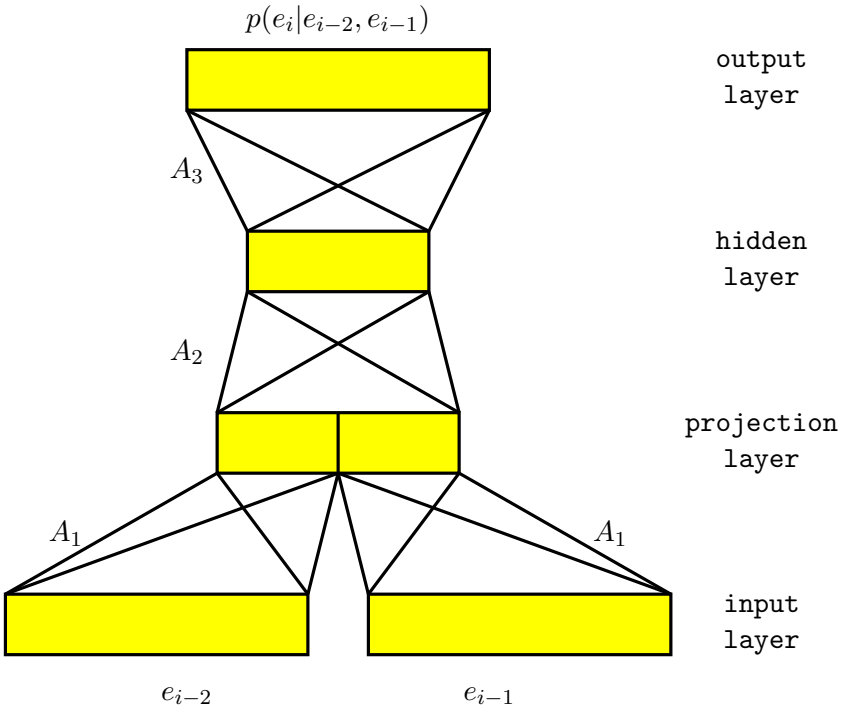


Figure 5.1: The architecture of a trigram feed-forward neural network language model. To compute the probability of e_i, the two preceding words, e_{i-1} and e_{i-2}, are used as inputs for the network.

5.3.2 Translation Models

For translation modeling, the neural network receives instead of the last n words some words from the source sentence as input. In the simplest case, it would receive one source word as input and should predict a possible target word as its translation. To know which word to use as input, this work uses the alignment created for phrase extraction, which is stored in the phrases and used during decoding to recreate the alignment between each target and source word.

Just one input word would most likely not outperform a count-based model since there would be no sparsity which would cause problems for the count-based model. The neural model could, therefore, not benefit from its strength. To leverage the neural network, we need more context as input. This can be done by including a window around the source word that we want to translate, e.g., one word before and after the current position as depictured in Figure 5.2. This allows the network to learn from the context in which the source word is used to decide which translation should be preferred.

Assuming I, the target length, and b_1^I, the target to source alignment, mapping the target word index to the most relevant source word, are known. This is a valid assumption here since the phrase-based system provides both. If more than one alignment for a target word is given, methods like the affiliation technique proposed in [Devlin & Zbib$^+$ 14] can be used to reduce it to one. The model can then be formalized and simplified like this:

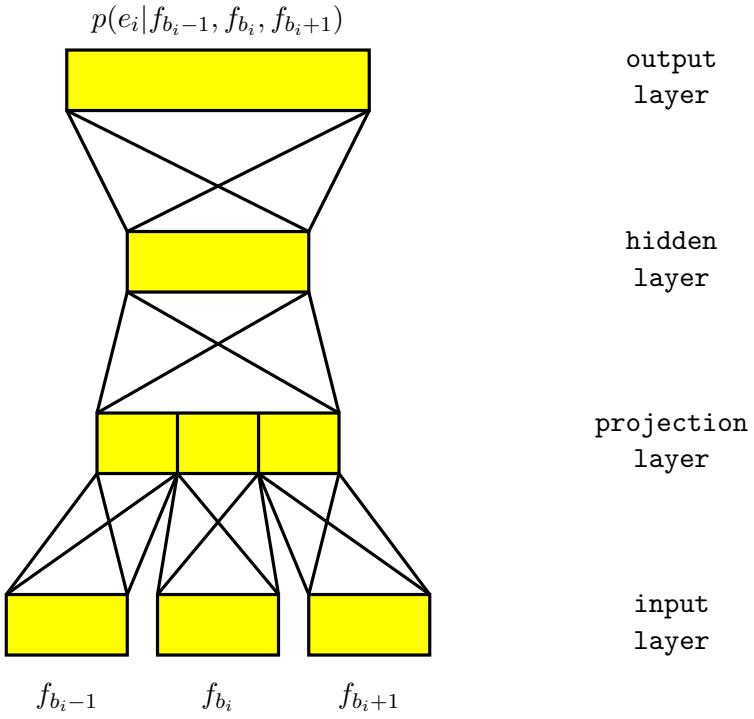


Figure 5.2: The architecture of a feed-forward neural network translation model with context window size 3. The aligned word (f_{b_i}) and one word in each direction (f_{b_i-1} and f_{b_i+1}) are the neural network input.

$$\begin{aligned} Pr(e_1^I|f_1^J) &= Pr(e_1^I|b_1^I, f_1^J, I) \\ &= \prod_{i=1}^{I} Pr(e_i|e_1^{i-1}, b_1^I, f_1^J, I) \\ &\approx \prod_{i=1}^{I} Pr(e_i|f_{b_i-\Delta_m}^{b_i+\Delta_m}) \end{aligned}$$

where $Pr(e_1^I|f_1^J)$ is the probability of the whole target sequence given the whole source sequence. $\Delta_m = \frac{m-1}{2}$ with m being the size of the window around the source word.

This model can also be computed in the opposite direction. While the reversed model cannot directly be used in decoding since the words after the current target position are not known, it can be added in rescoring:

$$\begin{aligned} Pr(f_1^J|e_1^I) &= Pr(f_1^J|a_1^J, e_1^I, J) \\ &= \prod_{j=1}^{J} Pr(f_i|f_1^{j-1}, a_1^J, e_1^I, J) \\ &\approx \prod_{j=1}^{J} Pr(f_j|e_{a_j-\Delta_m}^{a_j+\Delta_m}) \end{aligned}$$

where a_1^J is the source-to-target alignment.

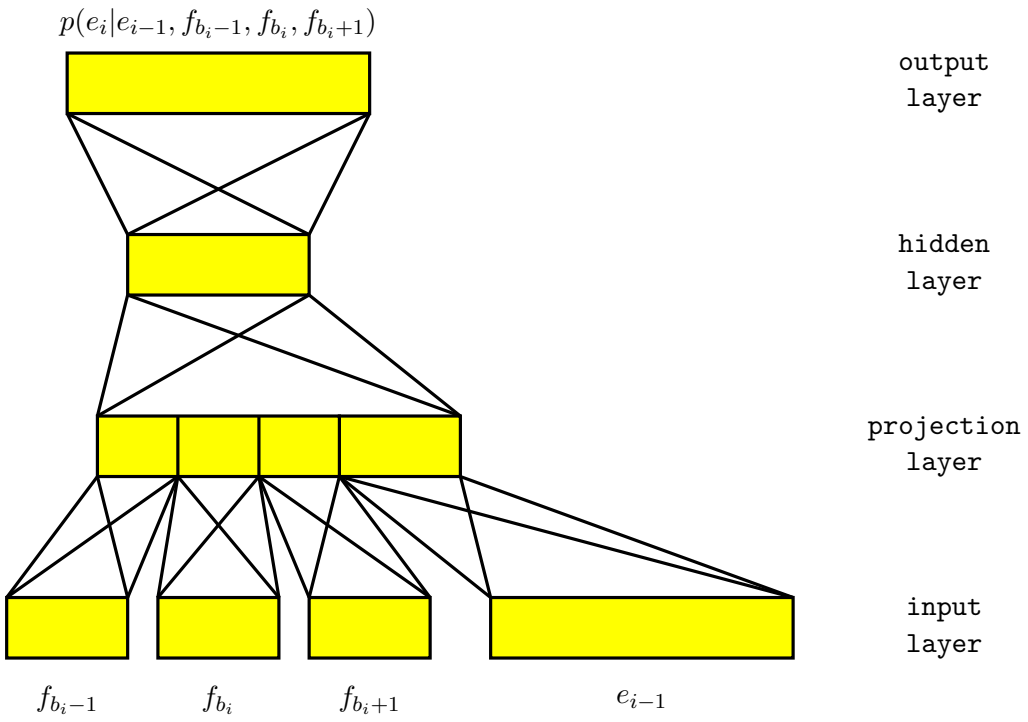


Figure 5.3: The architecture of a feed-forward neural network joint model with context window of size 3 and a bigram language model. Combining the information from the source and target side.

5.3.3 Joint Models

Combining the language model with the translation model gives us the joint model. This model allows us to integrate even more context information. If two words on the target side are aligned to the same word on the source side, a pure translation model cannot distinguish between these two cases since the input is identical. Including the language model information allows the model to learn the difference between both from the target side input.

We are assuming again that b_1^J and I are known similar to that translation model. The formal definition of the joint model is:

$$\begin{aligned} Pr(e_1^I|f_1^J) &= Pr(e_1^I|b_1^I, f_1^J, I) \\ &= \prod_{j=1}^{J} Pr(e_i|e_1^{i-1}, b_1^I, f_1^J, I) \\ &\approx \prod_{j=1}^{J} Pr(e_i|f_{b_i-\Delta_m}^{b_i+\Delta_m}, e_{i-n-1}^{i-1}) \end{aligned}$$

5.4 Bag-of-Words

Feed-forward neural network models, as described in the previous section, are limited in the capability to handle long-distance dependencies on source or target sides. The input of a language model if for example, limited to accept only $n-1$ previous words. Simply increasing n to an

arbitrarily large value will not solve this problem. The inputs that are further away from the current position are not adequately trained since the relevant words are with a greater distance from the predicted word, more likely to appear in different positions. This makes it much harder for the network to learn the relation between them.

While recurrent neural networks are designed to accept arbitrary long inputs, they need particular layers like the LSTM or GRU (Section 4.2.2) to avoid the vanishing gradient problem. [Mikolov & Joulin$^+$ 15] showed that using a *Bag-of-Words* (BoW) as additional input for a recurrent neural network-based language model avoids this problem. They achieved very similar perplexities on automatic speech recognition tasks in comparison to the LSTM neural network, whose structure is considerably more complicated. This suggests that the bag-of-words model can effectively store the long-distance contextual information, which could show improvements in statistical machine translation as well. Since the bag-of-words representation can cover as many contextual words without further modifying the network structure, the problem of only having a limited context window size of feed-forward neural networks is removed.

5.4.1 Bag-of-Words Vector

To represent a Bag-of-Words as a vector, we take a similar approach as [Irie & Schlüter$^+$ 15]. Instead of allowing only one value to be different to zero as with done with the one-hot vector, we allow multiple values to have a none zero value. Let us take these two sentences as an example:

1. `a man is at the door`
2. `the crocodiles snapped at the boat`

The n-hot vector for these sentences would be:

vocabulary:	a	at,	boat,	cat,	crocodiles,	door,	is,	man,	snapped,	the,	...
BOW_1:	(1,	1,	0,	0,	0,	1,	1,	1,	0,	1,	...)
BOW_2:	(0,	1,	1,	0,	1,	0,	0,	0,	1,	1,	...)

To avoid the effect that larger Bag-of-Words would cause stronger activation on the next layer than smaller bags, we place $1/n$, with n being the number of different words in the sentence, instead of 1. Not that we count each word only once. The resulting vector, similar to the input vectors for the models, now looks like this:

vocabulary:	a,	at,	boat,	cat,	crocodiles,	door,	is,	man,	snapped,	the,	...
BOW_1:	($\frac{1}{6}$,	$\frac{1}{6}$,	0,	0,	0,	$\frac{1}{6}$,	$\frac{1}{6}$,	$\frac{1}{6}$,	0,	$\frac{1}{6}$,	...)
BOW_2:	(0,	$\frac{1}{5}$,	$\frac{1}{5}$,	0,	$\frac{1}{5}$,	0,	0,	0,	$\frac{1}{5}$,	$\frac{1}{5}$,	...)

5.4.2 Definition

The bag-of-words model is a simplifying representation applied in natural language processing. In this model, each sentence is represented as the set of words disregarding the word order. Bag-of-words models are used in this work as additional input features for feed-forward neural networks in addition to the one-hot encoding. Thus, the probability of the feed-forward neural network translation model with an m-word source window can be written as:

$$Pr(e_1^I \mid f_1^J) \approx \prod_{i=1}^{I} Pr(e_i \mid f_{b_i-\Delta_m}^{b_i+\Delta_m}, f_{\text{BoW},b_i}) \tag{5.1}$$

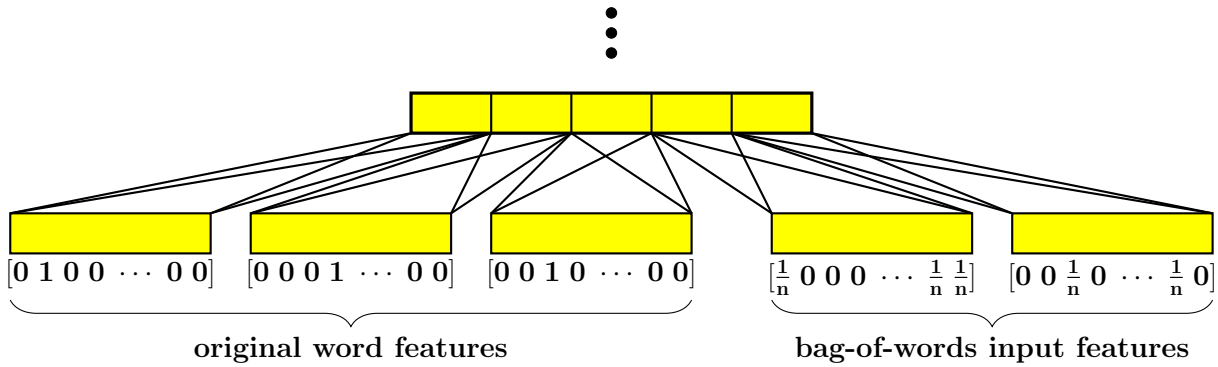


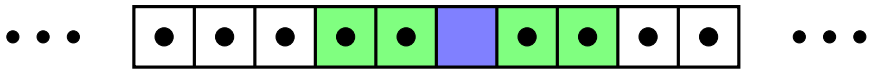
Figure 5.4: The bag-of-words input features, along with the original word features. The input vectors are projected and concatenated at the projection layer. We omit the hidden and output layers for simplification since they remain unchanged.

where $\Delta_m = \frac{m-1}{2}$ and b_i is the index of the single aligned source word to the target word e_i, similar to the translation model (Section 5.3.2). f_{BoW,b_i} is the bag-of-words for the position b_i. This additional input to a neural network is shown in Figure 5.4.

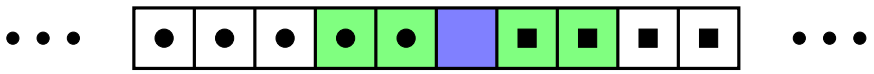
5.4.3 Contents of Bag-of-Words Features

Before utilizing the bag-of-words input features, we have to decide which words should be part of it. We tested multiple different variants:

1. Collecting all words of the sentence in one bag-of-words except the currently aligned word.

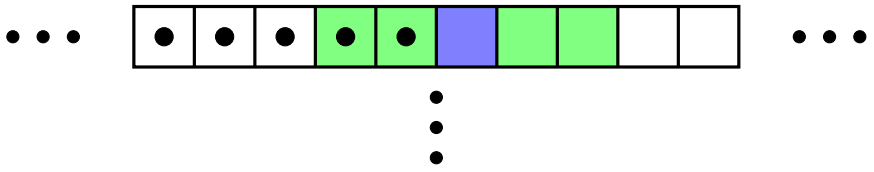


2. Collecting all preceding words in one bag-of-words and all succeeding words in a second bag-of-words.



3. Collecting all preceding words in one bag-of-words and all succeeding words in a second bag-of-words except those already included in the source window.

4. Only collecting the preceding words, which is also suitable for language models (Both [Mikolov & Joulin+ 15] and [Irie & Schlüter+ 15] applied this variant for their bag-of-words features):



All of these variants provide the feed-forward neural network with an unlimited context in both directions. The divergences between these setups only varied by 0.2% BLEU and 0.1% TER. We chose the third variant since it performed best and seemed to be the most logical choice for us.

5.4.4 Exponentially Decaying Bag-of-Words

Another variant is to weight the words within the bag-of-words model. In the standard bag-of-words representation, these weights are equally distributed for all words. This means that bag-of-words input is a vector which indicates if a word is given or not and does not encode the word order. To avoid this problem, the *exponential decay* approach proposed in [Clarkson & Robinson 97] has been adopted to express the distance of contextual words from the current word. Therefore the bag-of-words vector with decay weights can be defined as following:

$$\tilde{f}_{\mathrm{BoW},i} = \sum_{k \in S_{\mathrm{BoW}}} d^{|i-k|} \tilde{f}_k \tag{5.2}$$

where

i, k Positions of the current word and words within the BoW model respectively.

$\tilde{f}_{\mathrm{BoW},i}$ The value vector of the BoW input feature for the i-th word in the sentence.

$\tilde{f}_k$ One-hot encoded feature vector of the k-th word in the sentence.

S_{BoW} Indices set of the words contained in the BoW. If a word appears more than once in the BoW, the index of the nearest one to the current word will be selected.

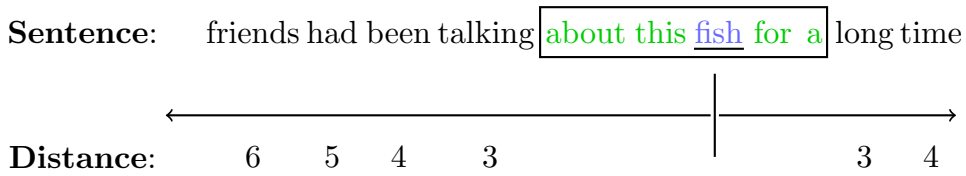
d Decay rate with float value ranging from zero to one. It specifies how fast the weights of contextual words decay along with the distances, which can be learned like other weight parameters of the neural network.

Instead of using a fixed decay rate as in [Irie & Schlüter$^+$ 15], we propose to train the decay rate like other weight parameters in the neural network. The approach presented by [Mikolov & Joulin$^+$ 15] is comparable to the corpus decay rate shown here, except that their work makes use of a diagonal matrix instead of a scalar as the decay rate. In our experiments, three different kinds of decay rates are trained and applied:

1. Corpus decay rate (Corpus DR): all words share the same decay rate.
2. Individual decay rate for each bag-of-words (BoW DR): each bag-of-words has its decay rate given the aligned word.
3. Individual decay rate for each word (Word DR): each word uses its own decay rate.

We use the English sentence "`friends had been talking about this fish for a long time`" as an example to clarify the divergences between these variants. A five words contextual window centered at the current aligned word `fish` has been applied: {`about`, `this`, `fish`, `for`, `a`}. The bag-of-words models are used to collect all other source words outside the context window: {`friends`, `had`, `been`, `talking`} and {`long`, `time`}.

There are multiple choices for assigning decay weights to all these words in the bag-of-words feature:



1. Corpus decay rate: d

   Weights: d^6 d^5 d^4 d^3 d^3 d^4

2. Bag-of-words individual decay rate: $d = d_{\text{fish}}$

   Weights: d^6_{fish} d^5_{fish} d^4_{fish} d^3_{fish} d^3_{fish} d^4_{fish}

3. Word individual decay rate: $d \in \{d_{\text{friends}}, d_{\text{had}}, d_{\text{been}}, d_{\text{talking}}, d_{\text{long}}, d_{\text{time}}\}$

   Weights: d^6_{friends} d^5_{had} d^4_{been} d^3_{talking} d^3_{long} d^4_{time}

It would also be possible to combine the last two variants, but since this would lead to a massive matrix with many, most likely, untrained entries, we did not test this setup.

5.5 Contributions

The author of this thesis implemented the feed-forward neural network framework and the models used for these experiments and combined it with the RWTH's opensource translation toolkit *Jane* in rescoring. Implementing the bag-of-words features was supervised in the scope of [Wang 15]. The bag-of-works features are published at ACL 2016 [Peter & Wang$^+$ 16].

6. Experimental Evaluation of Feed-Forward Models

This chapter evaluates the effect of different models and approaches presented in Chapter 4 on multiple tasks. This is done by first building a state-of-the-art phrase-based system and then adding these models to it.

6.1 Setup

The experiments are conducted on four different tasks that are chosen to represent a broad spectrum of different challenges.

6.1.1 German to English (WMT 2017)

As a well-known task of a European language pair, we chose the WMT 2017 German→English task [Bojar & Chatterjee+ 17]. What makes this language pair challenging is the long-distance reorderings that are needed during translation. Especially the position of the verb can change a lot. The German language also allows for compound words that increase the vocabulary, which creates an additional problem to handle the language. This task has about four million sentences a large amount of training data available. The corpora included as training data in this task are CommonCrawl, Europal, NewsCommentary, and all test sets between 2009 and 2013. Newstest2015 is used as a development set, newstest2014, newstest2016, and newstest2017 are used as test sets.

The 5-gram language model is trained on a total of 7.2 billion running words using the KenLM toolkit [Heafield 11, Heafield & Pouzyrevsky+ 13]. The training data of the language model is comprised out of the monolingual corpora Europarl v7, News Commentary v12, News Discussions v1 and v2, News Crawl 2007 through News Crawl 2016, as well as the target side of the bilingual data. Corpus statistics can be found in the Appendix, Table A.1.

6.1.2 Chinese to English (BOLT)

To represent an Asian language, we include the BOLT 2014 Mandarin Chinese→English task. This language pair is very challenging due to the vast differences between the two languages. This task contains about four million sentences, also a large amount of training data.

Two separate 5-gram language models are trained using the SRILM toolkit [Stolcke 02] on different subsets of the provided corpora, which are selected using the cross-entropy approach proposed by [Moore & Lewis 10]. Those models are then interpolated to obtain the final 5-gram language model. In total 2.9 billion running words are used for the final language model. We use dev12-tune as development set. The dev12-dev and P1R6-dev sets are used as test sets. Corpus statistics can be found in the Appendix, Table A.3.

6.1.3 English to Romanian (WMT 2016)

A task with a smaller amount of parallel data, around 0.6 million sentences, is the WMT 2016 English→Romanian task [Bojar & Chatterjee+ 16]. This task also represents translations out of English. Romanian is a language with a rich morphology, which makes it hard for machine translation systems to handle. We are using a 4-gram model trained on 7.5 billion running words using the KenLM toolkit for this task. The used data contains the target side of the bilingual data as well as all available monolingual data, namely the Common Crawl and the News Crawl 2015 corpora.

The provided development set was split into two to generate a test set, which was available to us before the final evaluation. We used the first part, newsdev2016.1, as our development set, while the second part, newsdev2016.2, and newstest2016 are used as test sets. Corpus statistics can be found in the Appendix, Table A.2.

6.1.4 German to English (IWSLT 2013)

A smaller number of experiments are done on the IWSLT 2013 German→English task [Cettolo & Niehues+ 13]. The baseline system is trained on all available parallel data, which is around four million sentences. We select parts of the Shuffled News and LDC English Gigaword corpora based on the cross-entropy difference [Moore & Lewis 10] as additional training data, besides the target side of the parallel data, for the 4-gram language model. A total of 1.7 billion running words are used for language model training. This baseline system in this task contains in addition to the regular language model also a 7-gram word cluster language model [Wuebker & Peitz+ 13b] and uses word reorderings of the source side as described in [Popović & Ney 06].

The neural networks for these tasks are only trained on the in-domain data, which contains about 159 thousand sentences. The parallel corpus statistics can be found in the Appendix, Table A.4.

6.1.5 Scoring

All experiments are evaluated using the Bleu (Section 3.3.2) and Ter (Section 3.3.1) metrics. The scores are computed as case-insensitive. The Bleu score is computed using the `mteval` script as it is distributed with `Moses` in version 13a [Koehn & Hoang+ 07]. The Ter score is computed with `TERCom` in version 0.7.25 [Snover & Dorr+ 06].

6.1.6 Integrating of Feed-Forward Networks in Phrase-Based Machine Translation

To evaluate the performance of the neural networks described in Chapter 5, we combine them with a separately trained phrase-based machine translation system. This can be done by using a rescoring step with n-best lists created by the system or by integrating it into decoding.

- Create n-best lists and add the new model score in a separate step. This enables the use of complicated models since the model is only required to score a very limited number of hypotheses. Additionally, the whole source and target side is known during the computation of the score. The only modification of the phrase-based system is to include a rescoring step and export all relevant information, like the word alignment, to the n-best lists.

- Integrating the neural model directly in decoding is the alternative. This avoids missing out on a good translation hypothesis, which receives a strong score from the new model but would not have made it into the n-best list.

We run experiments with both settings but used only translation models for the decoder integration, since everything can be precomputed at the beginning of the translation, causing only a small decrease in speed.

6.1.7 Phrase-Based System

GIZA++ [Och & Ney 03] is applied for aligning the parallel corpus. The scaling factors are tuned with MERT [Och 03] with BLEU as the optimization criterion on the development sets. We used the MERT run with the best BLEU score on the development set out of three. In the experiments, the maximum size of the n-best lists applied for rescoring is 500. For rescoring, we use 20 MERT processes with a different random seed and use the weights that performed best on the development set. Experiments are performed using the *Jane* toolkit [Vilar & Stein$^+$ 10, Wuebker & Huck$^+$ 12] with a log-linear framework containing following feature functions:

- Phrase translation probabilities in both directions
- Word lexicon features in both directions
- Enhanced low frequency counts [Chen & Kuhn$^+$ 11]
- 4-gram or 5-gram language model with modified Kneser-Ney smoothing [Kneser & Ney 95, Chen & Goodman 96]
- Word and phrase penalties
- Hierarchical reordering model [Galley & Manning 08]

6.1.8 Implementation

All here presented feed-forward neural network-based models where implemented from scratch using the `Theano` toolkit [Bergstra & Breuleux$^+$ 10]. They are trained using mini-batch gradient descent (Section 4.3.2). The *Jane* toolkit was modified to improve the alignment extraction and allow it to export that information to the n-best list output.

6.1.9 Neural Network Structure

The feed-forward neural network structure for the experiments in this chapter uses a projection layer with 100 nodes per input and two hidden layers, the first hidden layer with 1000 nodes and the second with 500 nodes. The output layer uses a short list containing the 10000 most frequent words. All other words are clustered into 1000 classes, and these classes are added to the output layer. So the total size of the output layer is 11000. Clustering all additional words into classes serves two purposes. It speeds up the computation of the output layer, and it ensures that all output nodes are seen frequently enough for the network to learn its meaning. Since we use the network to score the translation candidates provided by the phrase-based system, and not to predict them freely, we are not required to distinguish between all possible words and can use this class mapping. We choose these values after running multiple experiments. These values gave the best trade-off between performance and efficiency.

6.2 Baseline Models

We first test the effect of the language model, translation model, and joint model when added to a phrase-based machine translation system. The language model, as described in Section 5.3.1,

Table 6.1: Baseline experiments for Phrase-Based Machine Translation De-En

De-En (WMT 2017)	newstest2015		newstest2016		newstest2017	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
Baseline	30.3	52.2	34.4	47.7	30.0	52.6
+ Language Model	30.8	51.8	34.9	47.4	30.4	52.2
+ Translation Model	30.9	51.6	34.9	47.3	30.5	51.9
+ Joint Model	31.1	51.4	35.3	46.7	30.5	51.7

Table 6.2: Baseline experiments for Phrase-Based Machine Translation on En-Ro

En-Ro (WMT 2016)	newsdev2016/1		newsdev2016/2		newstest2016	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
Baseline	24.7	57.6	28.1	53.6	24.9	57.8
+ Language Model	25.4	56.9	28.4	53.4	25.4	57.4
+ Translation Model	25.5	56.7	28.8	52.6	25.5	56.9
+ Joint Model	25.6	56.6	28.8	52.9	25.6	57.0

uses, in these experiments, the previous five words on the target side as input while scoring the current word, making its inputs similar to a count based 6-gram language model. The translation model, as described in Section 5.3.2, uses a window of size five around the source word aligned to the current target word. Namely the aligned word, the two predecessor words, and the two successor words are used. Finally, the joint model, as described in Section 5.3.3, uses a combination of five predecessor words on the target side and a window of five on the source side.

WMT 2017 German to English (De-En)

The results for German→English are shown in Table 6.1. Adding the language model to the baseline system shows an improvement between 0.4% and 0.5% in BLEU as well as between 0.2% and 0.3% TER across all test sets. Using a translation model results in an improvement of 0.5% BLEU and between 0.4% and 0.5% TER compared to the baseline. The joint model performs best by improving 0.9% BLEU on newstest2016 and 1.0% TER.

WMT 2016 English to Romanian (En-Ro)

The translation and joint model behave similarly better than the language model on the English→Romanian task, as shown in Table 6.2. The difference between these models is less than 0.2% BLEU on newstest2016, while all models improve the baseline system between 0.5% and 0.7% BLEU on this task.

BOLT Chinese to English (Zh-En)

On the Chinese→English task, it is harder for the neural network model to show improvements (Table 6.3), especially on the dev12-dev set, where only the joint model improves slightly the baseline. We presume that this is due to the fact that the n-gram language model used here is highly tuned for this task, while the neural models used all available parallel data for training, which is from the general domain. For P1R6-dev, we see improvements similar to the English→Romanian task. The language model improves by 0.4%, the translation model by 0.5%, and the joint model by 0.6% BLEU. Note that, the translation model reaches a TER score better by 0.4% compared to

Table 6.3: Baseline experiments for Phrase-Based Machine Translation on Zh-EN

Zh-En (BOLT)	dev12-tune		dev12-dev		P1R6-dev	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
Baseline	20.3	65.6	18.6	67.1	17.3	67.3
+ Language Model	20.6	65.8	18.6	67.3	17.7	67.2
+ Translation Model	20.6	65.5	18.6	66.7	17.8	66.5
+ Joint Model	20.7	65.5	18.8	67.0	17.9	66.9

Table 6.4: Adding the bag-of-words input features without decay to the Chinese→English task

Zh-En (BOLT)	dev12-tune		dev12-dev		P1R6-dev	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
Baseline	20.3	65.6	18.6	67.1	17.3	67.3
+ Language Model	20.6	65.8	18.6	67.3	17.7	67.2
+ BoW Feature	20.6	65.8	18.7	67.2	17.8	67.3
+ Translation Model	20.6	65.5	18.6	66.7	17.8	66.5
+ BoW Feature	20.5	65.5	18.6	66.8	17.8	66.5
+ Joint Model	20.7	65.5	18.8	67.0	17.9	66.9
+ BoW Feature	20.8	65.5	18.7	67.1	17.7	66.9

the joint model. This is usually a sign for shorter translation for the system with a better score, since the TER metric prefers short translation, as it is also the case here. The hypothesis using the translation model reaches 99.0% of the reference length, while the hypothesis using the joint model reaches 99.5% of the length.

6.3 Bag-of-Words

In this section, we prove the effectiveness of the Bag-of-Words (BoW) approach presented in Section 5.4 and test the influence of different decay rates. We also compare it to feed-forward models with a larger source window and an LSTM model.

As input for the bag-of-words in case of a translation, we collected all words preceding in one bag-of-words and for the joint model all succeeding words in a second bag-of-words, except those that are already included in the source window as described in Section 5.4.3. The language model used only preceding words except the words that are already included as target side input for one bag-of-words, and the joint model received all three bag-of-words as input.

Simply adding the BoW input features to the model does not have a large effect on the BLEU score. Looking at the results on the Chinese→English task, Table 6.4, we see that the scores on some test sets are slightly improved while they decrease at the same time on others. Overall adding the BoW without the decay feature does not have a significant influence on the score.

6.3.1 Exponentially Decaying Bag-of-Words

Using a simple BoW as input leads to losing the information how far away the input is from the current position. This is giving words far away from the current position, the same impact as close words. As shown in Section 5.4.4, the exponential decay approach can be applied to express the distance of contextual words from the current word. This way, the information of sequence order can be included in bag-of-words models. We demonstrate the effect of three different kinds

Table 6.5: Bag-of-words experiments for Phrase-Based Machine Translation on De-En

De-En (WMT 2017)	newstest2015		newstest2016		newstest2017	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
Baseline	30.3	52.2	34.4	47.7	30.0	52.6
+ Language Model	30.8	51.8	34.9	47.4	30.4	52.2
+ BoW Feature	30.9	51.7	34.9	47.2	30.5	52.0
+ Corpus DR	30.9	51.6	34.8	47.3	30.5	52.1
+ BoW DR	31.0	51.8	35.1	47.2	30.6	52.0
+ Word DR	31.0	51.7	34.9	47.3	30.6	52.1
+ Translation Model	30.9	51.6	34.9	47.3	30.5	51.9
+ BoW Feature	31.0	51.2	35.1	46.8	30.7	51.6
+ Corpus DR	31.0	51.3	35.3	46.6	30.8	51.5
+ BoW DR	31.1	51.2	35.5	46.5	30.9	51.4
+ Word DR	31.1	51.3	35.5	46.4	30.7	51.3
+ Joint Model	31.1	51.4	35.3	46.7	30.5	51.7
+ BoW Feature	31.2	51.3	35.4	46.7	30.7	51.7
+ Corpus DR	31.3	51.3	35.4	46.7	30.7	51.6
+ BoW DR	31.4	51.1	35.6	46.6	31.0	51.5
+ Word DR	31.3	51.3	35.4	46.7	30.8	51.5

of decay rates for words in the bag-of-words input feature (BoW Feature), as described in Section 5.4.4:

1. the corpus decay rate (Corpus DR),
2. the bag-of-words individual decay rate (BoW DR),
3. and the word individual decay rate (Word DR).

WMT 2017 German to English

In Table 6.5 we see the results of all German→English WMT 2017 task experiments. Using the corpus decay rate does not differ significantly compared to the bag-of-words input without decay (Corpus DR) in any of the three models. If we make the decay rate depended on the word which defines the bag-of-words (BoW DR), the results show clear improvements across all models. Using the word individual decay rate (Word DR) shows similar improvements, but is overall a little behind the BoW DR . By using the bag-of-words with BoW DR , we can see an overall improvement between 0.2% BLEU, e.g., the language model on newstest2016, and 0.6% BLEU, the translation model also on newstest2016, across all models and test sets. The best language model improvement over the baseline we achieve is 0.7% BLEU and of 0.5 TER on newstest2016. At the same time, the translation model achieves improvements of 1.1% BLEU and 1.2% TER, and the joint model 1.2% BLEU and 1.1% TER using bag-of-words with the bag-of-words decay rate over the baseline on the same test set.

WMT 2016 English to Romanian

The Table 6.6 illustrates the scores for the English→Romanian task which give a similar picture. Bag-of-words decay rate shows consistent improvements with the word individual decay performing a little weaker. The corpus decay rate has little to no effect, at the same time.

Table 6.6: Bag-of-words experiments for Phrase-Based Machine Translation on En-Ro

En-Ro (WMT 2016)	newsdev2016/1		newsdev2016/2		newstest2016	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
Baseline	24.7	57.6	28.1	53.6	24.9	57.8
+ Language Model	25.4	56.9	28.4	53.4	25.4	57.4
+ BoW Feature	25.5	56.8	28.5	53.1	25.5	57.3
+ Corpus DR	25.6	56.7	28.5	53.1	25.6	57.1
+ BoW DR	25.6	56.8	28.7	53.0	25.6	57.2
+ Word DR	25.5	56.8	28.7	53.2	25.6	57.1
+ Translation Model	25.5	56.7	28.8	52.6	25.5	56.9
+ BoW Feature	25.5	56.2	28.9	52.1	25.5	56.5
+ Corpus DR	25.5	56.5	28.8	52.3	25.5	56.8
+ BoW DR	25.8	56.5	29.2	52.3	25.7	56.7
+ Word DR	25.7	56.5	29.0	52.5	25.7	56.7
+ Joint Model	25.6	56.6	28.8	52.9	25.6	57.0
+ BoW Feature	25.5	56.9	28.7	53.0	25.5	57.2
+ Corpus DR	25.4	57.1	28.7	52.7	25.4	57.1
+ BoW DR	25.8	56.4	29.1	52.4	25.9	56.8
+ Word DR	25.8	56.5	28.9	52.7	25.7	57.0

Adding the bag-of-words model with the right decay rate gives an improvement between 0.2% and 0.4% BLEU. The language model with BoW DR improves now by 0.6% and 0.7% BLEU over the baseline, again surpassed by the translation model, which achieves between 0.8% and 1.1% BLEU improvement. The joint model performs similar to the translation model on this task, with 1.0% BLEU improvement on both test sets.

BOLT Chinese to English

The experiments with the Chinese→English task show the smallest improvement of all, but the bag-of-words using BoW DR still shows clear improvements with all models and test sets, as shown in Table 6.7. The Word DR setup performs again on a very similar level as BoW DR . The joint model outperforms the translation model in this task by 0.2% BLEU, while the translation model is slightly stronger than the language model.

IWSLT 2013 German to English

For the IWSLT German→English 2013 task (Table 6.8), we compared the bag-of-words approach additionally with a long short-term memory recurrent neural network-based translation model [Sundermeyer & Alkhouli$^+$ 14]. The model contains three LSTM layers with 200 nodes each. The LSTM model improved over the baseline system by 1.5% BLEU and 1.7% TER on the test set. The feed-forward neural network-based translation model making use of a bag-of-words with BoW DR achieved a slightly better performance with an improvement of 1.7% BLEU and 2.1% TER.

Looking at all results, we see that the results of the word individual decay rate are slightly worse than that of the bag-of-words decay rate. One reason could be that in individual word case, the sequence order can still be lost. It also seems that the translation model profits most from the bag-of-words input and the language model the least.

All decay rates were initialized with 0.9 before tuning. As an example, in the IWSLT 2013

Table 6.7: Bag-of-words experiments for Phrase-Based Machine Translation on Zh-En

Zh-En (BOLT)	dev12-tune		dev12-dev		P1R6-dev	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
Baseline	20.3	65.6	18.6	67.1	17.3	67.3
+ Language Model	20.6	65.8	18.6	67.3	17.7	67.2
+ BoW Feature	20.6	65.8	18.7	67.2	17.8	67.3
+ Corpus DR	20.7	65.8	18.6	67.5	17.5	67.5
+ BoW DR	20.8	65.7	18.8	67.2	17.9	67.3
+ Word DR	20.7	65.8	18.8	67.1	17.9	67.2
+ Translation Model	20.6	65.5	18.6	66.7	17.8	66.5
+ BoW Feature	20.5	65.5	18.6	66.8	17.8	66.5
+ Corpus DR	20.6	65.3	18.8	66.5	17.9	66.3
+ BoW DR	20.7	65.4	18.9	66.5	18.0	66.5
+ Word DR	20.7	65.5	18.9	66.7	18.1	66.6
+ Joint Model	20.7	65.5	18.8	67.0	17.9	66.9
+ BoW Feature	20.8	65.5	18.7	67.1	17.7	66.9
+ Corpus DR	20.7	65.3	18.8	66.7	18.0	66.9
+ BoW DR	20.9	65.2	19.1	66.7	18.3	66.6
+ Word DR	20.8	65.3	19.1	66.6	18.2	66.5

Table 6.8: Experimental results for the IWSLT 2013 German→English task using exponentially decaying bag-of-words models with different kinds of decay rates.

De-En (IWSLT 2013)	test		eval11	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]
Baseline	30.7	49.1	36.0	44.0
+ Translation Model	31.9	47.5	36.7	43.0
+ BoW Feature	32.0	47.3	36.9	42.9
+ Corpus DR	32.1	47.3	36.9	42.7
+ BoW DR	32.4	47.0	37.2	42.4
+ Word DR	32.3	47.0	37.1	42.7
+ LSTM	32.2	47.4	37.1	42.5

German→English task, the value after tuning was 0.578. When investigating the values of the trained bag-of-words individual decay rate vector, we noticed that the variance of the value for frequent words is much lower than that for rare words. We also observed that most function words, such as prepositions and conjunctions, are assigned low decay rates. We could not find a pattern for the trained value vector of the word individual decay rates.

6.3.2 Comparison between Bag-of-Words and Large Context Window

The primary motivation behind the usage of the bag-of-words input features is to provide the model with additional context information. To refute the argument that merely increasing the size of the window could achieve the same results, we compared the bag-of-words input features with larger source side windows. Our experiments showed that increasing the source side window beyond 11 gave no further improvements, while the model that used the bag-of-words input features can achieve the best result. This is shown in Figure 6.1.

A possible explanation for this could be that the feed-forward neural network learns its input

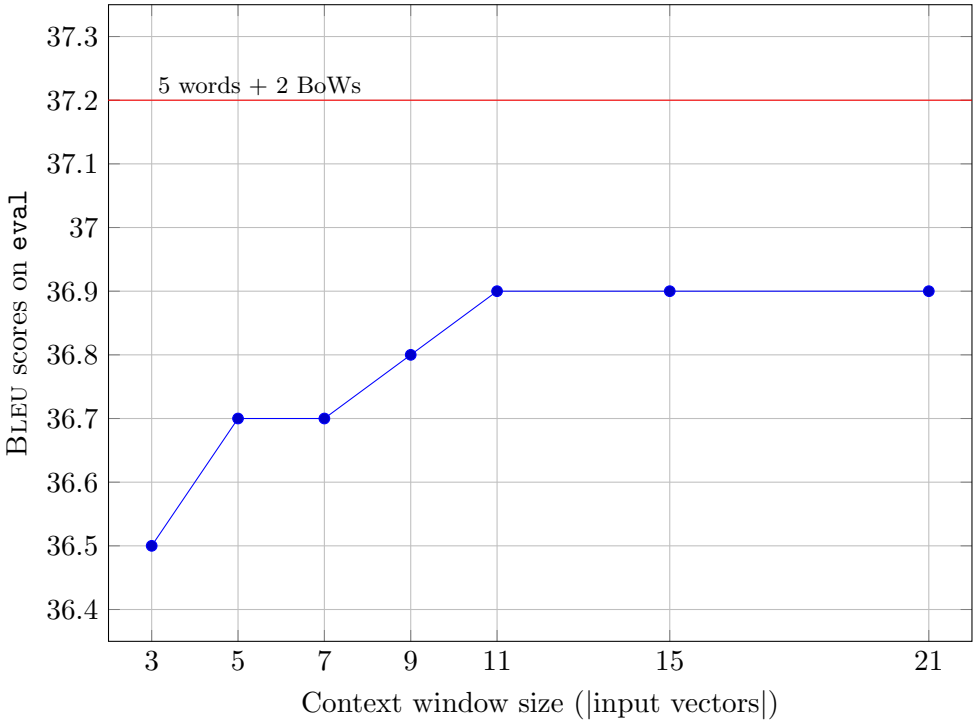


Figure 6.1: The change of BLEU scores on the eval set of the IWSLT 2013 German→English task along with the source context window size. The average sentence length of the corpus is about 18 words. The red line is the result of using a model with bag-of-words input features and a bag-of-words individual decay rate.

position-dependent. If a relevant context word is moved by just one position, it can turn a seen event into an unseen event. This is not a problem if enough similar events have been seen during training, but seeing these events gets less likely as the distance between the two words grows. At some point, the model will consider the signal too week and will not take these inputs into account. The bag-of-words model, on the other hand, still receives the same signal, only slightly stronger or weaker, depending on the new distance and the decay rate.

6.4 Neural Network Models in Decoding

Integrating the neural network joint model into the decoding of phrase-based systems, as proposed in [Devlin & Zbib$^+$ 14], requires a neural network optimized for fast response times to make a large number of predictions with different target contexts required during decoding. All predictions of the translation model, on the other hand, can be precomputed since they have no target dependencies. For each position on the source side, we compute and store the target vector, before we start the translation. We use, therefore, the translation models to evaluate how much improvement to expect when going from rescoring to decoding.

The results in Table 6.9 show that the integration into decoding yields most the time improvement of about 0.2% to 0.3% BLEU and TER. These improvements are in line with the findings reported in [Alkhouli & Rietig$^+$ 15], which also involved language models. We would expect a similar improvement of the previous experiments in this chapter, if all neural networks were to be fully integrated into decoding.

Table 6.9: Decoding experiments for Phrase-Based Machine Translation on the IWSLT 2013 De-En task

De-En (IWSLT 2013)	test		eval	
	BLEU [%]	TER [%]	BLEU [%]	TER [%]
Baseline	30.7	49.1	36.0	44.0
+ Translation Model in Rescoring	31.9	47.4	36.7	42.7
+ BoW Feature	32.0	47.2	36.9	42.6
+ BoW DR	32.4	46.7	37.2	42.1
+ Translation Model in Decoding	32.2	47.1	37.0	42.5
+ BoW Feature	32.3	47.0	37.2	42.4
+ BoW DR	32.5	46.7	37.4	42.0

6.5 Conclusion

In this chapter, we have performed an experimental analysis of the methods presented in Chapter 5. We show the effectiveness of adding language models, translation models, or joint models to three highly optimized state-of-the-art phrase-based systems. Joint models show the most substantial improvement, shortly followed by translation models. The pure language models still improve the baseline system in nearly all cases, but it is the weakest model of these three.

As next step we presented the results of adding a bag-of-words input to the neural network models. The simple version, which losses information about word order, does not show any relevant gains, and adding a decay rate for the whole corpus was only marginally better. By training a separate decay rate for each word that defines the bag-of-words, we show that consistent improvements of about 0.3% BLEU are possible. This method allows the model to weight how important the inputs, which are further way, are for the specific word that is currently translated. A separate decay rate for individual word shows slightly weaker, but still consistent improvements. Note that all models show improvements by adding the bag-of-words input with a separate decay rate for each word that defines the bag-of-words. However, the translation models show overall the most gains, while the language models gain the least.

Comparing the bag-of-words input to feed-forward translation models with larger input windows shows that merely increasing the input window does not achieve the performance of the bag-of-words input. The comparison with the LSTM recurrence translation model on the German→English IWSLT 2013 task shows that both models perform similarly well, with a slight advantage for the bag-of-words model. The LSTM model improves the baseline system by 1.1% BLEU and 1.5% TER while the bag-of-words model improved it by 1.2% BLEU and 1.6% TER.

In the last section of this chapter, we compare adding the translation models in rescoring and decoding. We see that decoding outperforms the implementation in rescoring between 0.1% and 0.3% BLEU. Using only the translation model allows us to precompute all translation model scores before decoding.

While the improvements gained with the bag-of-words models are not vast, they are still consistent. We used these models and methods extensively to reach strong placements in the international WMT and IWSLT evaluation campaigns.

6.6 Contributions

The author of this thesis wrote the complete feed-forward neural network implementation used as the basis for these experiments. He also implemented the changes needed in the RWTH's

open-source translation toolkit *Jane* to use these models in rescoring. The bag-of-words features as well as integrating the translation models into decoding was done by Weiyue Wang and supervised by the author in the scope of [Wang 15]. The author conducted all experiments on the German→English WMT 2017, English→Romanian WMT 2016, and Chinese→English Bolt tasks. The experiments on the German→English IWSLT 2013 task were conducted by Weiyue Wang and supervised by the author. Jörn Wübker provided the LSTM model used for comparison.

7. Encoder-Decoder Models with Attention

The previous chapters present methods how a phrase-based system can be improved by adding a neural network in the log-linear model. It is possible to show clear improvements this way, but the neural network can only score translations proposed by the phrase-based system. While this limitation gives more control over the created translations, it also limits the possible improvements that can be achieved with neural networks. This chapter focuses on a machine translation model only use neural networks and modifications to it. We start with a short introduction into neural machine translation in Section 7.1 and an overview of related works in Section 7.2. The encoder-decoder approach with attention is presented in Section 7.3, followed by Section 7.4 explaining different modifications of the attention computation. In Section 7.5 we present two modifications to the neural network training process, the guided alignment training (Section 7.5.1) and curriculum learning (Section 7.5.2). To obtain a word alignment, like it is used in guided alignment training, we present in Section 7.6 a method on how to modify the neural network to produce alignments that outperform state-of-the-art systems. The end of this chapter is Section 7.7 with a detailed list of who contributed to this chapter.

7.1 Introduction

Neural Machine Translation (NMT) has been introduced in 2014 and was quickly adopted by the community. This approach uses either a combination of convolutional [Kalchbrenner & Grefenstette$^+$ 14, Gehring & Auli$^+$ 17b, Gehring & Auli$^+$ 17a], recurrent [Bahdanau & Cho$^+$ 15, Sutskever & Vinyals$^+$ 14, Cho & van Merriënboer$^+$ 14a], feed-forward [Vaswani & Shazeer$^+$ 17], or a combination of both [Chen & Firat$^+$ 18] network structures to map a sequence of arbitrary length into another sequence of arbitrary length. The whole process, from parsing the source sequence to scoring the target sequence, is here modeled inside of a neural network.

7.2 Related Work

Based on the NMT approach by [Bahdanau & Cho$^+$ 15], researchers have tried to improve the translation quality by modifying the attention mechanism. Most methods add various features to the attention computation [Tu & Lu$^+$ 16, Mi & Sankaran$^+$ 16, Sankaran & Mi$^+$ 16, Feng & Liu$^+$ 16, Cohn & Hoang$^+$ 16, Yang & Hu$^+$ 17, Zhang & Wang$^+$ 17]. Others attempt to change the attention mechanism itself [Zhang & Xiong$^+$ 16, Meng & Lu$^+$ 16] or the network structure [Wang & Lu$^+$ 16].

Research related to change the neural network training similar to Section 7.5 has been done in both areas. External alignments have been utilized to teach the network to mimic those by adding the alignments to the objective function [Chen & Matusov$^+$ 16, Mi & Wang$^+$ 16a] or by

adding an alignment model to the decoding process [Alkhouli & Bretschner+ 18, Alkhouli & Ney 17].

The work of [Bengio & Louradour+ 09] present curriculum learning for neural networks, and [Kocmi & Bojar 17] presents it for neural machine translation. At the same time, we know that the local sentence order can be changed to improve the efficiency of the training without hurting the network performance [Khomenko & Shyshkov+ 17, Doetsch & Golik+ 17].

We also looked at work that analysis the attention and tries to build alignments similar to the target foresight approach (Section 7.6). A method to create alignments using posterior regularization was presented by [Ganchev & Graça+ 10] and [Tamura & Watanabe+ 14], which used a special-purpose recurrent neural network to create alignments. Recently [Garg & Peitz+ 19] presented a method to create alignments using transformer models.

To the best of our knowledge, there have been these publications that empirically measure the impact of their approach on the alignment quality [Tu & Lu+ 16, Mi & Sankaran+ 16, Mi & Wang+ 16a, Sankaran & Mi+ 16]. These investigations use the Saer [Tu & Lu+ 16], Aer [Och & Ney 03] and F_1 metrics to measure the alignment quality. All authors noticed an improvement in alignment quality by applying their extensions to the attention mechanism. However, as [Mi & Wang+ 16a] report, there is still a significant qualitative difference to state-of-the-art alignments.

7.3 Encoder-Decoder Model

The encoder-decoder architecture, as presented in [Sutskever & Vinyals+ 14] uses the encoder to encode the source sentence in a fixed-sized vector and the decoder to create the target sentence given that vector. This architecture was then extended by adding an attention mechanism [Bahdanau & Cho+ 15] to improve the performance on long sentences. We will now explain this model in more detail since the experiments in Chapter 8 are all based on this architecture.

7.3.1 Encoder

The encoder consists out of one or more recurrent layers which are given the representations of each token of the source sentence as input. This representation is again trained using one-hot vectors similar to the projection layer in feed-forward networks 4.1.1:

$$E_f : \mathbb{R}^{|V_f|} \rightarrow \mathbb{R}^{d_f}$$
$$E_f(\tilde{f}) = W_f \tilde{f}$$

where $\tilde{f}$ is the one-hot vector associated with f. All neural network models in this work with recurrence use either LSTM [Hochreiter & Schmidhuber 97] or GRU layers [Cho & van Merrienboer+ 14b]. We write this recurrence operation using the ϕ symbol:

$$y_i = \phi(x_i, y_{i-1})$$

where x_i is the input from the previous layer, and y_{i-1} is the recurrent input from the previous time step. The initial state, y_0, is usually trained as part of the model, alternatively it is set to contain only zeros. The encoder input sequence f_1^J is encoded from left to right as follows:

$$\overrightarrow{h_j} = \phi(E_f(\tilde{f}_i), h_{j-1})$$

for $j = 1, \ldots, J$. This can also be done in the reverse direction starting at the last word:

$$\overleftarrow{h_j} = \phi(E_f(\tilde{f}_i), h_{j+1})$$

Do combine the information from both directions at each position we can also use a bidirectional encoder:

$$h_j = \overrightarrow{h_j} \circ \overleftarrow{h_j}$$

The bidirectional encoding allows the network to have access to the whole source sentence for all positions.

7.3.2 Decoder

The decoder produces the target sentence given the encoded source sentence. There are different variants of how the source sentence can be encoded. These different variants are either using:

- the sentence end of the forward encoder: $\overrightarrow{h_J}$,
- the sentence start of the reverse encoder: $\overleftarrow{h_1}$,
- a combination of both: $\overrightarrow{h_J} \circ \overleftarrow{h_1}$, or
- a normalized sum over all positions: $\frac{1}{J}\sum_j h_j$

This state is then either used to initialize the hidden state s_0 [Sutskever & Vinyals$^+$ 14]:

$$s_i = \phi(E_e(\tilde{e}_{i-1}), s_{i-1})$$
$$p(e_i|e_1^{i-1}, f_1^J) = softmax(Ws_i + b)|_{e_i}$$

or is provided again as input to the decoder at every position [Cho & van Merrienboer$^+$ 14b]:

$$s_i = \phi(E_e(\tilde{e}_{i-1}), s_{i-1} + h_J)$$
$$p(e_i|e_1^{i-1}, f_1^J) = softmax(Ws_i + b)|_{e_i}$$

where $\text{softmax}(x)|_i$ is the softmax value at position i, and $\tilde{e}_{i-1}$ is the one-hot vector of the target word at position $i-1$.

[Sutskever & Vinyals$^+$ 14] reported that the network got better results if the reversed encoder was used compared to the forward encoder. Using the reversed encoder gives a short distance between the beginning of the source sentence and the target sentence while increasing the distance between the last words in a sentence. This gives the impression that the network has problems transporting the information from the beginning of the source sentence to the end to initialize the decoder.

7.3.3 Attention

Since only one vector is passed from encoder to decoder, all relevant information has to be encoded. A sentence can be arbitrarily long and can, therefore, contain an arbitrarily large amount of information. It does not seem to be plausible to reduce every possible source sentence to a fixed-sized vector. Experiments also support this since the performance of the pure encoder-decoder architecture degrades with longer sentences [Bahdanau & Cho$^+$ 15]. This effect can be significantly reduced by adding an attention layer. The attention layer allows the neural decoder to focus on particular areas on the encoder side [Bahdanau & Cho$^+$ 15, Luong & Pham$^+$ 15] for

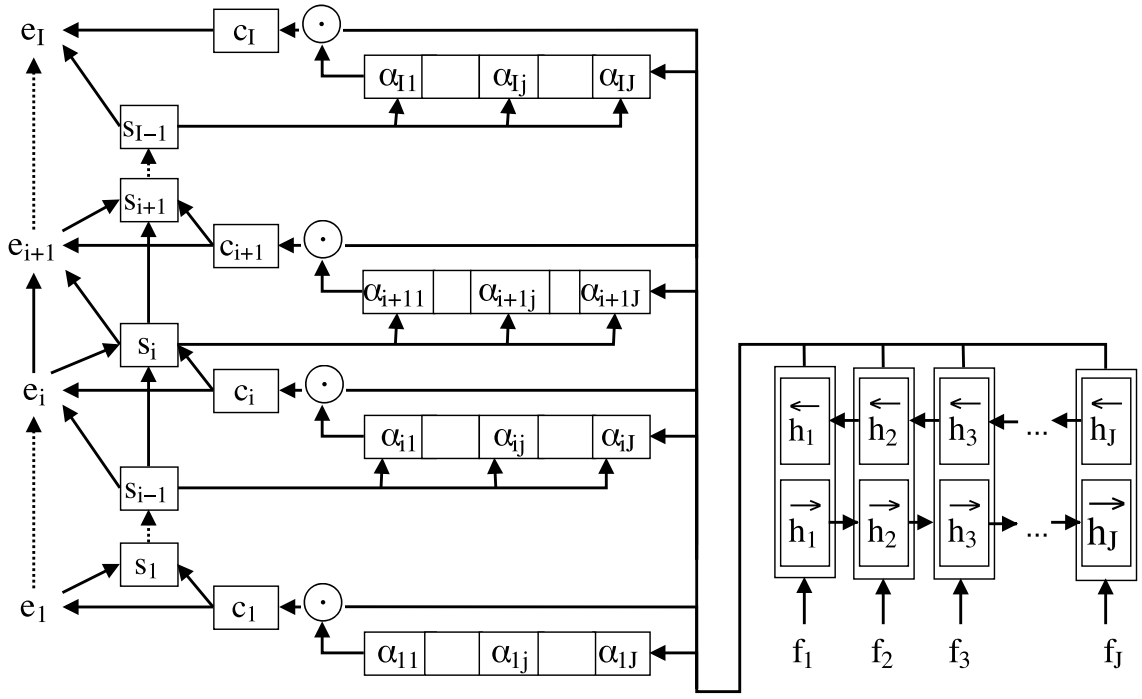


Figure 7.1: This is a simplified representation of the attention-based neural machine translation model, as presented in [Bahdanau & Cho$^+$ 15]. On the right side is the encoder with its hidden states h_1 to h_J. On the left side is the decoder with the attention mechanism. Using the α_{ij}, where i represents the position on the decoder side and j the position on the encoder side, these states are weighted and merged ($\odot$) the context vector c_i. In the last step, this context vector is then used to compute the output word e_i and the next state of the decoder s_i.

each time step. For each decoder state, it computes a weighted average over all encoder states. This allows bypassing the problem of squeezing all information into one vector and allows for a shorter distance between the decoder state and the relevant encoder state. This idea is shown in Figure 7.1.

The decoder starts by initialing the states of y_0, s_0 using one of the representations, as shown in Section 7.3.2. In this thesis, we are using the combination of the backward and forward encoding. As the next step, the attention is computed, the default method used in this work if not state otherwise is the additive attention mechanism as proposed by [Bahdanau & Cho$^+$ 15]:

$$r_{ij} = v^T \tanh(W_s s_{i-1} + W_h h_j + b_r) \tag{7.1}$$

$$r_i = (r_{i1}, r_{i2}, \ldots, r_{ij}, \ldots, r_{iJ}) \tag{7.2}$$

$$\alpha_{ij} = softmax(r_i)|_j \tag{7.3}$$

where v^T, W_s, W_h, and b_r are part of the model. These α values, from now on called attention weights, are used to compute a weighted average over the hidden states of the encoder, the context vector c_i:

$$c_i = \sum_{j=1}^{J} \alpha_{ij} h_j$$

This is used for two purposes. The first usage is to compute the probability distribution for the next output:

$$y_i = W_r(s_{i-1} \circ E_e(\hat{e}_{i-1}) \circ c_i)$$
$$p(e_i|e_1^{i-1}, f_1^J) = softmax(W_y y_i + b_y)|_{e_i}$$

where W_r, W_y, and b_y are part of the model. The second usage is to update the hidden state of the decoder:

$$s_i = \phi(E_e(\hat{e}_{i-1}) \circ c_i, s_{i-1})$$

This approach allows the network to learn how to choose the attention weights and focus on the part of the source sentence, which is most relevant for the current decoder position.

7.3.4 Search

The decoding in neural machine translation uses a simple beam search, compared to the phrase-based system. This search starts by generating for the first target position all possible words. The resulting list is, in the next step, reduced to the chosen beam size. To achieve this reduction, all translation candidates are sorted, and only as many as the beam size allows are kept. All possible next words are created for each of these translation candidates, and the resulting list is again reduced. We repeat this loop until all possible candidates in the search list produced an end of sentence symbol, or we reach a predefined maximum length limit. The score is normalized with respect to its length and the candidate with the best score is used as the final translation.

Modifications like limiting the vocabulary [Mi & Wang$^+$ 16b, Hu & Lan$^+$ 15] or more dynamic pruning strategy [Wu & Schuster$^+$ 16, Freitag & Al-Onaizan 17] have shown to speed up the search.

7.4 Modifications of the Attention Computation

One shortcoming of the described attention computation in Section 7.3.3 is that there is no direct way to store which source positions are already covered. It has to be stored indirectly inside the hidden state. If storing this information fails, the model might try to access the same source word several times. Experiments have shown that attention models sometimes end up creating a loop over the same few words over and over again. In this section, we describe modifications to the alignment computation to add a direct connection between the current attention computation and previous precomputed attentions, as shown in Figure 7.2.

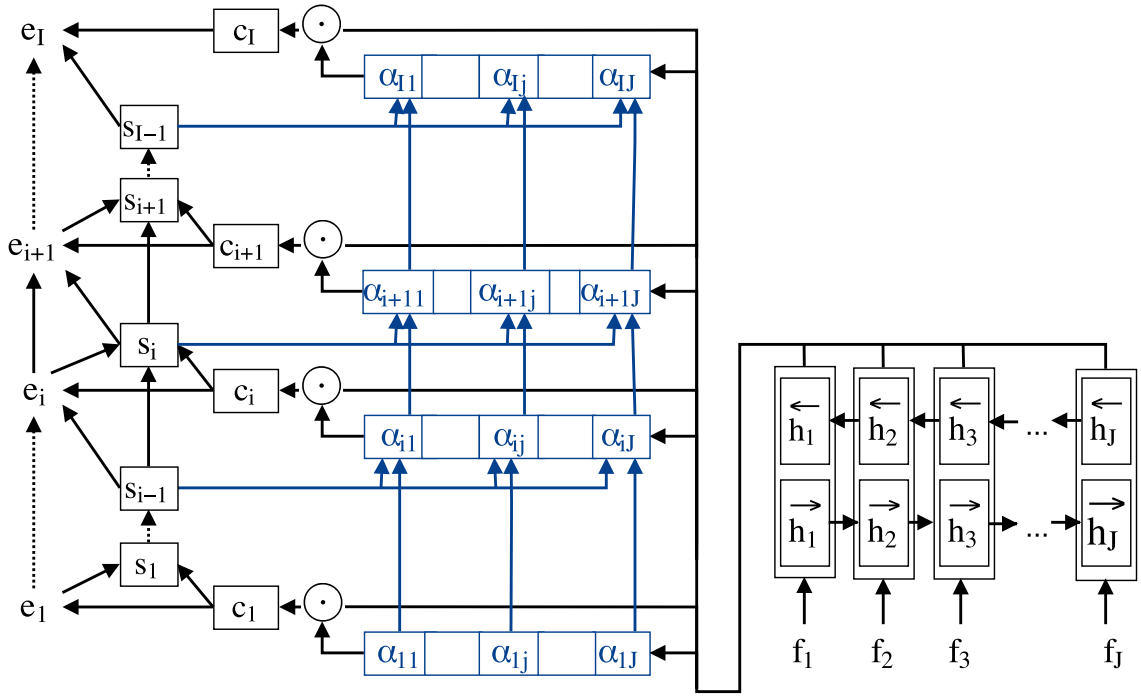


Figure 7.2: The modifications presented in Section 7.4 aim to add a direct connection between the current attention computation and previously computed attentions, as shown here.

7.4.1 Alignment Penalty

A simple solution to counter this effect of placing the attention repeatedly at the same positions, is to keep track of all alignments of the past decoder steps and sum them up in a variable $\beta_{i,j}$. For each decoder step, we compute the following sum:

$$\beta_{ij} = \sum_{k=1}^{i-1} \alpha_{kj} \tag{7.4}$$

β_{ij} stores how intensively each encoder state was already used during the decoding process. This information is then added as a penalty to the computation of the alignment energies. By doing so, we give the encoder states that have already been used a lower weight.

$$r'_{ij} = r_{ij} - q\beta_{ij}$$

The factor q is a scalar used as weight for the penalty. It is initialized with 1 and updated during training. A fixed weight should not be used at this point since r is an unnormalized value with an unbound range. Learning the factor as part of the model allows the network to find the best for it.

7.4.2 Alignment Feedback

While the alignment penalty is a fast and straightforward method, it has the drawback that all words share the same penalty. The encoder or decoder state is not used to adjust the penalty.

The alignment computation could be modified at Equation 7.1 to add additional features to the attention computation. These features can also include pieces of information about the encoder and decoder state. One way to achieve this would be:

$$r_{ij} = v^T \tanh(W_s s_{i-1} + W_h h_j + b_r + W_\beta \beta_{ij}) \tag{7.5}$$

A similar approach is mentioned in [Tu & Lu$^+$ 16]. W_β is a weight matrix of size $m \times 1$ where m is the "match dimension". β_{ij} is similar to the definition in Equation 7.4. We do not limit the annotation to a single alignment weight but also allow a window around j. Now the equation becomes:

$$r_{ij} = v^T \tanh(W_s s_{i-1} + W_h h_j + b_r + W_\beta [\beta_i]_{j-k}^{j+k})$$

The window size is defined by k. This window allows us to include the knowledge of the surrounding alignments into the computation. Including this modification before v^T and tanh allows for nonlinear behavior and gives the network more freedom to include the previous alignment information into the computation.

Fertility

The IBM Model 3 [Brown & Della Pietra$^+$ 93] uses fertility to determine how many target words should be produced by a single source word. The work by [Cohn & Hoang$^+$ 16] and [Tu & Lu$^+$ 16] show how this concept can be integrated into neural networks using alignment feedback.

The alignment penalty and the alignment feedback $\beta_{i,j}$ is defined as a sum over all α_j until the decoder step i. To add fertility, we can divide β by the expected fertility. e.g., if a single word should have the fertility of two, we would divide β by two. It would, therefore, take $2 \cdot \beta$ to reach the same penalty as a word with the fertility of one would have. Based on Equation 7.5 we are now defining β_{ij} as:

$$\beta_{ij} = \frac{1}{\Phi_{h_j}} \sum_{k=1}^{i} \alpha_{kj}$$

The fertility does depend on the encoder state instead of using the source word embedding directly. This allows having different fertilities for the same word but within a different context. The fertility Φ_{h_j} is defined as:

$$\Phi_{h_j} = N\sigma(v_\Phi^T h_j)$$

Where N is the maximum fertility for a word, which is set externally, and v is a vector transforming the hidden state into a scalar value. v is learned as part of the model. We can now interpret β_i as coverage vector in step i since it is modeling how much of the source sentence is already covered.

7.4.3 Alignment History

Besides feeding the model the information about previous alignments at the current position, it is also possible to provide the network with the knowledge of how well the previous alignment positions are already covered. This should help the network to avoid having larger gaps in the source coverage.

$$\gamma_{i,j} = \frac{\sum_{k=1}^{j-1} \beta_{i,k}}{j-1}$$

This additional input is then added to the attention computation (Equation 7.1):

$$r_{ij} = v^T \tanh(W_s s_{i-1} + W_h h_j + b_r + W_\gamma \gamma_{ij}) \tag{7.6}$$

where W_γ is the weight matrix for γ.

7.4.4 Convolutional Alignment Feedback

Convolutional alignment feedback uses the hybrid attention mechanism proposed by [Chorowski & Bahdanau$^+$ 15] to adapt the attention-based approach to speech recognition. Let us define the feedback component γ_i as a one-dimensional convolutional operation over the attention weights α_i:

$$\gamma_i = W\alpha_i$$

with $W \in \mathbb{R}^{M\times(2k+1)}$. We extract M vectors and apply for each vector a different filter W_m. Every filter W_m uses a window of size $2k+1$ which is centered around position j.

$$\gamma_{m,ij} = \sum_{l=j-k}^{j+k} W_{m,j-l} \cdot \alpha_{il} \cdot \alpha_i \;\forall\; m = 1, \ldots, M$$

The additional input is then added to the attention computation (Equation 7.1), similar to the alignment history in Equation 7.6. This modification allows the network to consider the attention which was used in previous time steps when computing the current attention weights.

7.5 Training Modifications

Instead of changing the model, as done in Section 7.4, we present in this section two approaches to modify the model training.

7.5.1 Guided Alignment Training

The attention can be interpreted as a probabilistic notion of an alignment distribution. In the attention-based model, it is trained together with the other parts of the network. During training, the network is guided to the optimal values using the cost function $\mathcal{L}$, which is defined on the network outputs. We are using the conditional log-likelihood of the target sentence e_1^I given f_1^J. The cross-entropy for a corpus of size N is computed as follows:

$$\mathcal{L}_{\text{ce}} = -\frac{1}{N}\sum_{n=1}^{N} \log p_\theta((e_n)_1^{I_n}|(f_n)_1^{J_n}) \tag{7.7}$$

Where p_θ is the modeled probability distribution with parameter set θ. I_n and J_n are the number of words in target sentence $(e_n)_1^J$ and the matching source sentence $(f_n)_1^J$.

Minimizing the decoder error does not give us any direct influence on the internal attention computation. Since we are not evaluating the performance of the system on the decoder cost, but on translation quality, improving the attention training might help in terms of overall performance. Additionally, given the complexity of the network, guiding the attention from beginning in the right direction simplifies the training task. Otherwise, the network needs to learn this information from a weaker signal and might get stuck in a local minimum.

To guide the training, we introduce a second cost function that gives direct feedback to that attention mechanism using separately computed alignments, similar to [Chen & Matusov$^+$ 16] and [Mi & Wang$^+$ 16a]. We compute this loss function as the cross-entropy between a given target alignment A_{ij} and the attention α_{ij}, which is computed by the network and is interpreted as a soft alignment.

$$\mathcal{L}_{\text{align}} = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{I_n}\sum_{j=1}^{J_n} A_{n,ij}\log\alpha_{n,ij} \tag{7.8}$$

This alignment loss function (Equation 7.8) is then combined with the original loss function (Equation 7.7) using a weighted sum to build the new loss function for the network:

$$\mathcal{L} := \lambda_{\text{ce}}\cdot\mathcal{L}_{\text{ce}} + \lambda_{\text{align}}\cdot\mathcal{L}_{\text{align}}$$

the weight factors λ_{ce} and λ_{align} need to be set externally. Using this new loss function, we can use the target alignment to influence the training directly.

The word alignment needed for this method is generally not available, and handcrafted alignments are expensive to create. We, therefore, have to rely on automatic methods to obtain the word alignment. Tools like GIZA++ [Och & Ney 03] can be used to create these alignments for us. These alignments need to be normalized over the source words to make them compatible with the constraints imposed by the cross-entropy cost function:

$$A_{ij} = \frac{\tilde{A}_{ij}}{\sum_{k=q}^{J}\tilde{A}_{ik}} \;\forall\; i = 1,\ldots,I$$

7.5.2 Curriculum Learning

If a human learns a new language, he would most likely start with easy short sentences and increase the length and complexity over time. The training samples for neural networks are, in contrast, usually presented in random order. We try two variants of how to simplify the learning task for the neural network. The first variant is to use a word alignment to extract shorter phrases from the training data and use this data to initialize the training. This allows concentrating on learning the relation between fewer words, which should be more natural than being presented with a whole sentence.

In the second approach, we assume that the short sentences are simple examples and present them before the most likely more challenging, longer sentences. To achieve this, we separate the training data into groups. We start with over representing the shorter sentences and adding more and more longer sentences gradually from the other baskets to the training.

7.6 Target Foresight

The attention mechanism produces a distribution over the source sentence for every decoding step. This distribution is often interpreted as a soft alignment between the source and target

sentences. It has been shown that incorporating alignment information during training as an additional objective function can improve the overall performance of the system [Chen & Matusov+ 16]. This indicates that the alignment problem is still relevant.

The relation between attention and alignments motivates an approach, which aims at using the attention-based neural network model to generate word alignments. The attention mechanism has a disadvantage compared to conventional word alignment methods. While the word alignment is computed including the knowledge of the whole source and target sentence, the neural network knows only previously seen words on the target side. To remove this disadvantage, we extend the standard attention computation by introducing knowledge of the target word to which we want to align.

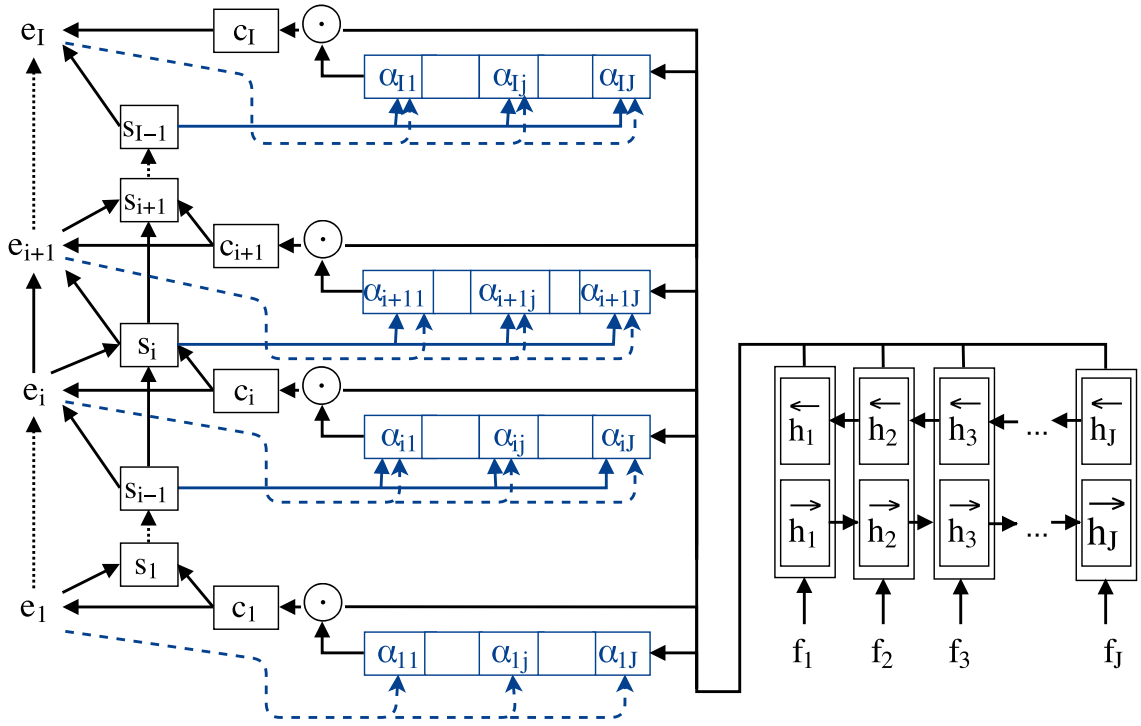


Figure 7.3: Attention-based NMT with target foresight, the dotted lines show how the current target word is fed back to the alignment computation.

7.6.1 Limitations of the Attention Mechanism

Equation 7.1 shows how the attention, without normalization, in the neural machine translation approach, as described in Section 7.3 is computed:

$$r_{ij} = v^T \tanh(W_s s_{i-1} + W_h h_j + b_r) \tag{7.1}$$

Since the network cannot know which word will be produced next the attention will focus on the position on the source side, it considers the most relevant to predict the next word. If the target side is given and the chosen word differs from the network's expectation the chosen position is most like irrelevant for the correct target word.

This limitation is not a problem if we create a fresh translation since the neural network scores decide which word will be created. These words will most likely correspond to the given attention. It becomes a problem if we do not aim for a direct translation, but to create a word alignment as it is used in Section 7.5.1.

7.6.2 Attention Modification

We aim to modify the attention mechanism to create alignments that are optimized for neural networks. The latter is essential since the attention mechanism does not assign weights to the source words, but to the encoder representation that is generated from these words. This representation may consequently encode information about neighboring words in the source sentence.

Nevertheless, we interpret the attention weights as a soft alignment and try to improve the alignment quality compared to the standard attention mechanism. We follow the example of traditional alignment methods and use the knowledge of the target reference sentence e_1^I to improve the alignment quality of the attention. Therefore, we introduce the embedding of the target word at the current decoding steps $\tilde{e}_i$ as additional input for the attention energy computation:

$$r_{ij} = v^T \tanh(W_s s_{i-1} + W_h h_j + W_e \tilde{e}_i + b_r) \tag{7.1}$$

We refer to this approach as *target foresight* (TF), since the network is allowed to use the foresight of the target word embedding $\tilde{e}_i$ to determine the corresponding source position that should be aligned to e_i. Figure 7.3 shows the additional connection added to the neural machine translation model.

To further investigate the target foresight approach, we propose three different methods to be applied during training. First, we add random noise to the value of $\tilde{\alpha}_{ij}$, which is supposed to prevent the encoding of target-word information in the attention weights. The second approach is to freeze the values of all weight matrices except for the attention parameters in the update steps of the training. The last approach is to train target foresight using guided alignment training [Chen & Matusov$^+$ 16, Mi & Wang$^+$ 16a]. The idea behind this approach is to motivate the network not to diverge too far from a given alignment. It allows, however, to choose a different alignment point if the improvement in the translation cost is significant enough.

7.7 Contributions

The implementation of most attention modifications, namely Alignment Penalty (Section 7.4.1), Alignment Feedback (Section 7.4.2), Fertility (Section 7.4.2), and Alignment History (Section 7.4.3), was done in the scope of Nick Rossenbach's bachelor thesis [Rossenbach 16] which was motivated and closely supervised by the author.

Arne Nix implemented the Convolutional Alignment Feedback (Section 7.4.4) as additional attention modification. The guided alignment training idea was proposed by Evgeny Matusov and Wenhu Chen and refined in personal conversations with the author, who is a co-author of [Chen & Matusov$^+$ 16]. The here described implementation was done by Arne Nix (Section 7.5.1).

It was the idea of the author to modify the neural network to use it to create alignments using the target foresight (Section 7.6), which was then implemented by Arne Nix [Peter & Nix$^+$ 17]. These three implementations are part of Arne Nix's bachelor [Nix 16] thesis and which was motivated and closely supervised by the author.

The author motivated the idea for curriculum learning as described in Section 7.5.2. Henrik Rosendahl implemented it as part of his bachelor thesis [Rosendahl 16] and was closely supervised by the author.

8. Experimental Evaluation of Encoder-Decoder Models

In this chapter, we show the experimental results for the attention-based models and modifications presented in Chapter 7.

After explaining the basic setup in Section 8.1, we start in Section 8.2 to test the effect of the attention modification proposed in 7.4. As next step, we show in Section 8.3.1 how the guided alignment approach performs in four different tasks, and in Section 8.3.2 the curriculum learning approach. Section 8.4 shows how we can modify the attention-based encoder-decoder model to create state-of-the-art alignments. As the last part of the experiments, we analyze in Section 8.5 some properties of the beam search.

The chapter finishes with the conclusion (Section 8.6) on the contributions (Section 8.7).

8.1 Setup

The experiments are conducted on the same tasks used for the phrase-based experiments described in Section 6.1, namely WMT 2017 German→English (Section 6.1.1), WMT 2016 English→Romanian (Section 6.1.3), BOLT Chinese→English (Section 6.1.2), and IWSLT 2013 German→English (Section 6.1.4).

8.1.1 Baseline

We use a word-embedding size of 620 for the projection layer, and a 30K shortlist of the most frequent words. The decoder and both directed RNNs of the bi-directional encoder are implemented as LSTM units. These RNNs, as well as the attention layer, have an internal dimension of 1000 nodes. For decoding, we use a beam-size of 12. We use a shortlist of 30k words, and for each unknown word, we copy the word where the attention points from the source side to the target side [Jean & Cho$^+$ 15, Luong & Sutskever$^+$ 15]. While we are aware of approaches like subword units [Sennrich & Haddow$^+$ 15] or sentence pieces [Kudo & Richardson 18] to avoid the problem of limiting the vocabulary, we decided to use the shortlist to keep the experiment setup as close as possible to the phrase-based system.

If not specified otherwise, we use Adam (Section 4.3.6) to train the neural network for 500k batches with a batch size of 75 sentences per batch. The model is saved and evaluated on the development set after every 10k batches. To improve the model stability and performance, we average the four snapshots that perform best on the development set as proposed in [Junczys-Dowmunt & Dwojak$^+$ 16]. For decoding, we use a beam-size of 12. Our implementation of the encoder-decoder model with attention is based on the Blocks framework [Van Merriënboer & Bahdanau$^+$ 15] and the deep-learning library Theano [Bergstra & Breuleux$^+$ 10].

Comparing the baseline results of the neural machine translation approach to the full phrase-based approach in Table 8.1.1 and Table 8.2 shows a drop in performance measured in Bleu and

Table 8.1: This table shows a comparison between the full phrase-based system, the phrase-based system using a language model limited to parallel data, and the baseline NMT system for German→English. The full phrase-based system outperforms the NMT system by a large margin. However, if both systems are limited to the parallel data, the NMT baseline noticeably outperforms the phrase-based system.

De-En (WMT 2017)	newstest2015		newstest2016		newstest2017	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
Phrase-Based full LM	30.5	52.0	34.6	47.5	30.2	52.4
Phrase-Based parallel LM	26.3	55.2	29.3	51.5	26.1	55.5
NMT Baseline	27.1	52.7	31.7	48.2	26.7	53.2

Table 8.2: This table shows a comparison of the full phrase-based system, the phrase-based system using a language model limited to parallel data, and the baseline NMT system for English→Romanian. We see that the NMT system outperforms the phrase-based system by a large margin, if both are limited to the parallel data. If the phrase-based system uses the monolingual data it performs similar to the NMT system.

En-Ro (WMT 2016)	newsdev2016/1		newsdev2016/2		newstest2016	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
Phrase-Based full LM	24.7	57.6	28.1	53.6	24.9	57.8
Phrase-Based parallel LM	21.1	60.7	22.9	58.4	21.4	61.1
NMT Baseline	24.7	56.3	27.2	53.2	25.4	56.1

TER of the NMT system. This can be explained by the fact that the model used here requires parallel data for training, which reduces the amount of data that can be directly used by a large margin. While there are methods to make monolingual data usable [Sennrich & Haddow+ 16], we used only the given parallel data for training to make the experiments easier to reproduce. Not using the monolingual data as well as not using domain adaptation explains the drop in performance of these baseline systems compared to the En-Ro and En-De systems in Chapter 6. The models used there utilize a language model trained on additional monolingual data and domain adaptation.

8.2 Modifications of the Attention Computation

In this section, we show the experimental results of the different modifications we made to the attention mechanism, as presented in Section 7.4.

8.2.1 Alignment Penalty

Adding the most straightforward method we presented to directly integrate previous attentions into the attention computation, the alignment penalty (Section 7.4.1), we see mixed results, as shown in Table 8.3. There is a small improvement on the WMT 2017 German→English task of up to 0.3% BLEU. On the WMT 2016 English→Romanian the BLEU score increases by 0.3% BLEU on the development set newsdev2016/2 while at the same time it decreases on our test set newsdev2016/1 by 0.2% BLEU. We see a loss in BLEU on the BOLT Chinese→English task for all test sets.

Table 8.3: Comparison of baseline model and model with alignment penalty.

De-En (WMT 2017)	newstest2015		newstest2016		newstest2017	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	27.1	52.7	31.7	48.2	26.7	53.2
+ Alignment Penalty	27.3	52.6	31.9	47.9	27.0	52.9
En-Ro (WMT 2016)	newsdev2016/1		newsdev2016/2		newstest2016	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	24.7	56.3	27.2	53.2	25.4	56.1
+ Alignment Penalty	24.5	56.4	27.5	52.6	25.4	56.3
Zh-En (BOLT)	dev12-tune		dev12-dev		P1R6-dev	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	19.4	66.6	18.4	67.1	17.8	66.8
+ Alignment Penalty	19.3	66.0	18.1	66.9	17.7	66.3

Table 8.4: Comparison of baseline model and model with alignment penalty trained using AdaDelta.

Zh-En (BOLT)	dev12-tune		dev12-dev		P1R6-dev	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	17.3	70.3	16.6	71.3	16.2	70.8
+ Alignment Penalty	18.1	69.1	17.0	69.6	16.9	69.5

In experiments on the same data but using AdaDelta (Section 4.3.5) which we conducted before using Adam we see a more favorable result for the alignment penalty. While having a weaker baseline with AdaDelta we see, for example, improvement of up to 0.8% BLEU on the BOLT Chinese→English task (Table 8.4). It seems that, using Adam, the network is able to learn the simple information provided by the alignment penalty another way.

8.2.2 Alignment Feedback with Fertility

The experimental results of the alignment feedback with fertility as presented in Section 7.4.2 are shown in Table 8.5. Using this method, we find small improvements on all tasks. WMT 2017 improves by 0.4% BLEU on the newstest2017 test set and even shows a minimal improvement on newstest2016. We see stagnation on the WMT 2016 English→Romanian development set (newsdev2016/1) and the newstest2016 test set, but an increase of 0.3% BLEU on the newsdev2016/2 test set. On the BOLT Chinese→English task, we find a consistent improvement of exactly 0.2% BLEU on all three test sets, but also a small decrease in TER in two cases. While the difference to the baseline is not large, it does improve on most test sets and languages.

8.2.3 Alignment History

The results of our experiments with adding the alignment history into the attention computation as presented in Section 7.4.3 are shown in Table 8.6. We see diverse results again. On WMT 2017 German→English all numbers improve, especially on newstest2017, where we see a gain of 0.6% BLEU. The results on WMT 2017 English→Romanian are mixed with gains of 0.4% BLEU on newsdev2016/2 and losses of 0.3% BLEU on newstest2016. BOLT Chinese→English shows the

Table 8.5: Comparison of baseline model and model with alignment feedback and fertility.

De-En (WMT 2017)	newstest2015		newstest2016		newstest2017	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	27.1	52.7	31.7	48.2	26.7	53.2
+ Alignment Feedback	27.4	52.7	31.8	48.0	27.1	52.9
En-Ro (WMT 2016)	newsdev2016/1		newsdev2016/2		newstest2016	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	24.7	56.3	27.2	53.2	25.4	56.1
+ Alignment Feedback	24.7	56.2	27.5	52.5	25.4	56.0
Zh-En (BOLT)	dev12-tune		dev12-dev		P1R6-dev	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	19.4	66.6	18.4	67.1	17.8	66.8
+ Alignment Feedback	19.6	66.9	18.6	66.5	18.0	67.2

Table 8.6: Comparison of baseline model and model with alignment history.

De-En (WMT 2017)	newstest2015		newstest2016		newstest2017	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	27.1	52.7	31.7	48.2	26.7	53.2
+ Alignment History	27.3	52.5	31.8	48.1	27.3	52.7
En-Ro (WMT 2016)	newsdev2016/1		newsdev2016/2		newstest2016	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	24.7	56.3	27.2	53.2	25.4	56.1
+ Alignment History	24.9	55.9	27.6	52.3	25.1	56.4
Zh-En (BOLT)	dev12-tune		dev12-dev		P1R6-dev	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	19.4	66.6	18.4	67.1	17.8	66.8
+ Alignment History	18.7	65.6	18.3	66.8	17.0	67.1

worst results with results worse than the baseline on all test sets. Overall, this method does not seem to improve the baseline system.

8.2.4 Convolutional Alignment Feedback

Table 8.7 shows the experimental results of the Convolutional Alignment Feedback. m and k are defined as described in Section 7.4.4. In short, m gives the number of filters used, and k defines the window size. If k=2 means that two positions in each direction are covered in addition to the current position, this will cover a total of five positions.

The numbers shown for WMT 2017 German→English indicate that the convolution alignment feedback works best with the settings k=5 and m=2. While the feedback does not have much effect on newstest2015 and newstest2016 it improves by 0.5% BLEU on newstest2017. Both other settings perform a little weaker, the setting with one large filter (k=10, m=1) even falls behind the baseline on the dev set (newstest2015). The setting (k=5, m=2), performs also the strongest on WMT 2017 English→Romanian. It grains 0.8% BLEU on newsdev2016/2 and 0.3% BLEU on newstest2016. The other two settings are similar to the German→English task very close to the baseline with (k=2, m=5) being the weaker setting. On the BOLT Chinese→English task,

Table 8.7: This table presents the results of the Convolutional Alignment Feedback. m defines how many filters are used, and k defines the window size.

De-En (WMT 2017)	newstest2015		newstest2016		newstest2017	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	27.1	52.7	31.7	48.2	26.7	53.2
+ convolution (k=2, m=5)	27.0	52.9	31.8	48.2	26.8	53.6
+ convolution (k=5, m=2)	27.1	52.8	31.8	48.0	27.2	53.0
+ convolution (k=10, m=1)	26.8	53.0	31.7	48.4	27.0	53.2
En-Ro (WMT 2016)	newsdev2016/1		newsdev2016/2		newstest2016	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	24.7	56.3	27.2	53.2	25.4	56.1
+ convolution (k=2, m=5)	24.7	56.4	27.5	52.4	25.3	56.0
+ convolution (k=5, m=2)	24.7	56.2	28.0	52.3	25.7	56.3
+ convolution (k=10, m=1)	25.0	55.8	27.4	52.6	25.5	55.9
Zh-En (BOLT)	dev12-tune		dev12-dev		P1R6-dev	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	19.4	66.6	18.4	67.1	17.8	66.8
+ convolution (k=2, m=5)	19.6	65.5	18.4	66.8	18.1	66.2
+ convolution (k=5, m=2)	19.6	66.0	18.5	66.6	17.9	66.1
+ convolution (k=10, m=1)	19.3	67.0	18.3	66.4	17.6	67.2

we see only little change by adding the convolution alignment feedback. While the setting with (k=10, m=1) falls slightly behind the baseline, the others slightly outperform it. The (k=2, m=5) setting improves by up to 0.3% BLEU while the (k=5, m=2) setting is more stable with a small improvement in all cases.

While the overall improvements are small, the setting with (k=2, m=5) is consistently able to at least reach the performance of the baseline. In most cases even outperform it.

8.3 Training Modifications

We present here the results of keeping the model fixed and modifying the training procedure as presented in Section 7.5.

8.3.1 Guided Alignment Training

The guided alignment training presented in Section 7.5.1 uses a separately computed word alignment of the parallel training data as additional input. This word alignment for these experiments is computed using the GIZA++ toolkit [Och & Ney 03]. We use the IBM 1, HMM, and IBM 4 models with five EM iterations per model. The resulting alignments in both directions are combined using the grow-diagonal-final-and heuristic [Koehn & Och$^+$ 03].

In Table 8.8 we see improvements of between 0.2% and 0.6% BLEU by using this method on the WMT 2017 German→English task. On the WMT 2016 English→Romanian task, the improvements are even larger with at least 0.4% and up to 0.8% BLEU. While the method works well on the first two tasks, it shows only minimal improvements on the BOLT Chinese→English task. We ran the experiment on that task, also with AdaDelta, which shows the same behavior. It seems that the GIZA++ alignment is not helpful for the Chinese→English task. We assume

Table 8.8: Results of the guided alignment training experiments.

De-En (WMT 2017)	newstest2015		newstest2016		newstest2017	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	27.1	52.7	31.7	48.2	26.7	53.2
+ Guided Alignment	27.3	52.7	32.2	47.6	27.3	52.7
En-Ro (WMT 2016)	newsdev2016/1		newsdev2016/2		newstest2016	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	24.7	56.3	27.2	53.2	25.4	56.1
+ Guided Alignment	25.4	55.6	27.9	52.3	25.8	55.4
Zh-En (BOLT)	dev12-tune		dev12-dev		P1R6-dev	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	19.4	66.6	18.4	67.1	17.8	66.8
+ Guided Alignment	19.4	66.0	18.5	66.4	17.9	66.5

Table 8.9: Guided alignment experiments on IWSLT2013 De-En with AER computed on the Europarl Gold Alignment corpus [Vilar & Popovic+ 06].

De-En (IWSLT 2013)	dev		test		Europarl	
Model	BLEU[%]	TER[%]	BLEU[%]	TER[%]	AER[%]	SAER[%]
Attention-Based	30.5	48.7	29.3	50.6	41.8	66.3
+ guided alignment	31.5	47.2	30.3	49.0	35.4	44.2

that for this language pair the difference between the optimal alignment for the NMT system and the alignment created by GIZA++ is too big.

For the IWSLT 2013 German→English task, we evaluate additionally the alignment quality of our models. To do this, we use a set of 504 bilingual sentence pairs that were extracted from the Europarl [Koehn 05] German→English task and manually aligned by human annotators [Vilar & Popovic+ 06]. We use this test set to evaluate the alignment quality on AER [Och & Ney 03] and SAER [Tu & Lu+ 16]. To evaluate the soft alignment with AER, we convert it into a hard alignment by extracting the position with the largest alignment weight in both directions, and merge them by applying Och's grow-diagonal-final-and heuristic [Och & Ney 03]. The baseline and the guided alignment model are both trained using AdaDelta.

The results presented in Table 8.9 show that the improvements are on the best test sets 1.0% BLEU and that the alignment error rate drops by 6.4%. This shows that the model incorporates the given knowledge of the externally provided alignment to compute the attention more similar to the alignment.

8.3.2 Curriculum Learning

To test the effect of curriculum learning (Section 7.5.2) we tested preferring short sentences at the beginning of the training and extracting short phrases as additional training data.

For the length ordered training, we divided the data into short sentences with up to 10 words on each side and sentences with more words. In the IWSLT 2013 data, around 0.5M sentences (10.7% of the total data) have less than ten words on each side, and 4M sentences have more words (89.3%). During length ordered training, we oversample these short sentences at the beginning to make up 50% of the training data and linearly decrease it to be equal to the real distribution

Table 8.10: Curriculum experiments on IWSLT 2013 German→English task. l gives the length of the added phrases and r the ratio of phrases of the total training data at the beginning.

De-En (IWSLT 2013)	dev		test		eval11	
	BLEU[%]	TER[%]	BLEU[%]	TER[%]	BLEU[%]	TER[%]
NMT Baseline	30.9	47.3	29.4	49.3	33.9	44.4
+ phrases (l=2-4, r=50%)	30.4	47.6	28.8	49.3	33.9	44.0
+ phrases (l=2-4, r=99%)	30.8	47.1	29.2	49.5	33.9	44.5
+ phrases (l=1-4, r=99%)	30.7	46.9	29.1	49.3	33.5	44.8
+ length ordered training	30.0	48.5	28.5	50.3	32.5	45.5
+ phrases (l=2-4, r=50%)	31.1	46.9	29.8	48.4	33.5	44.4
+ phrases (l=2-4, r=99%)	31.3	47.2	29.2	49.2	34.3	44.6

after 200k batches.

The additional phrases are extracted from the training data after the data are aligned using the GIZA++ toolkit. We extracted phrases with up to four words on each side using the same definition of a phrase as described in Section 3.2.6. We start with a given ratio of short phrases and linearly reduce the number of phrases until they are completely phased out at batch 200k.

We see in Table 8.10 that adding the extracted phrases to the training data in most cases slightly degrades the results. The results for the length ordered training shows even a large drop of up to 1.5% BLEU. We got more positive results by combining both methods. The picture on the other test sets is not so clear, while the settings with r=99% improve on test by 0.4% BLEU it loses on the eval11 set by the same amount. Setting r=50% gives the reversed result. The methods tested here did not show positive results and more experiments are needed to find a setup which improves the training reliably.

8.4 Target Foresight

To evaluate the effectiveness of our approach presented in Section 7.6 we compare it to GIZA++ [Och & Ney 03], the BerkeleyAligner[1], `fast_align` [Dyer & Chahuneau+ 13], and an unmodified attention-based model.

8.4.1 Setup

We use a mostly similar setup for this task as the described baseline model in Section 8.1.1, except that we used gated recurrent units instead of LSTMs in our recurrent layers. To evaluate the alignment quality of our models, we use the same test set as described in Section 8.3.1.

The network was trained on the Europarl corpus [Koehn 05], excluding the test set using AdaDelta [Zeiler 12] for learning rate adaption. Excluding the test data is done to evaluate the performance of the attention-based model on unseen data as is the case when used for translation. It also shows that target foresight can easily be used to align unseen data without the need to retrain the model, while still outperforming traditional methods that have been trained, including the test data. The training data consists of 1.2 million bilingual sentences of 32 million German and 34 million English running words. The training is performed for 250K iterations with a batch size of 40 and evaluated every 10K iterations. The development set of the IWSLT2013

[1] https://code.google.com/archive/p/berkeleyaligner

Table 8.11: Comparison of target foresight with the pure attention-based approach (with and without guided alignment) and other alignment methods on IWSLT 2013 German→English. The methods used in fast_align, GIZA++, and BerkeleyAligner are all using knowledge of the target side to compute the alignment similarly to target foresight.

	Alignment Test	
Model	AER %	SAER %
fast_align	27.9	33.0
GIZA++	21.0	26.8
BerkeleyAligner	20.5	26.4
Attention-Based	38.1	63.6
+ Guided alignment	29.8	38.0
+ Target foresight with fixed en-/decoder	33.9	55.6
+ Target foresight with guided alignment	**19.0**	34.9
+ converted to hard alignment	**19.0**	**24.6**

German→English shared translation task is used to select the best performing model which is then evaluated on the IWSLT2013 test as well as on the Europarl alignment test set.

8.4.2 Alignment Evaluation

Table 8.11 shows that GIZA++ creates a far better alignment than `fast_align` and that the BerkeleyAligner creates an even slightly better result. In comparison, the attention mechanism produces an AER of 17.6% worse than the BerkeleyAligner.

When we are interpreting the attention of the attention-based approach as an alignment result, we get an AER of 38.1%. When we train the network using guided alignment, we can reduce the AER to 29.8%.

Using the target foresight directly to create an alignment produces no usable results. The network does not learn any meaningful alignment but uses the attention weights to encode the target word $\hat{e}_i$. It is in nearly all cases able to reproduce the target word on the output layer, even though $\hat{e}_i$ is only given to compute the alignment. Furthermore, the computed alignment has no meaningful correlation with the correct alignment. To prevent this behavior, we try to make it harder to encode the target word into the attention weights, by applying noise to the alignment weights and the outputs of the corresponding network components. We also tried to initialize the encoder and decoder using the weights from our trained baseline network. We omitted these numbers since none of these techniques gave usable results and used the following methods instead.

Fixing the encoder and decoder weights of our baseline network and training the attention layer for just an additional 2000 iterations results in an improvement of 4.2% AER and 8.0% SAER.

Pairing the guided alignment training with the target foresight training yields an AER of 19.0%. This is an improvement of 10.8% compared to only using guided alignment. Compared to the BerkeleyAligner it improved by 1.5% and by 2.0% compared to GIZA++. Note the latter two still perform better considering the SAER score.

An explanation for this behavior is that the design of SAER makes it easier for systems with hard-alignments to perform better than a system using soft-alignments. To elaborate on this point, even if the soft and the hard-alignment create the correct alignment, the soft-alignment will most likely receive a lower score, since it is nearly impossible for the soft-max output layer to predicts the correct point with 100% certainty. Most alignment points are predicted correctly

Table 8.12: The effect of the beam size on decoding time, BLEU score and model cost on the IWSLT 2013 German→English task. The search time was measured on a CPU machine and reported as factor how much longer it takes compared with the beam search with beam 1.

De-En	search time	dev			test		eval11	
beam	≈ factor	BLEU[%]	TER[%]	avg. cost	BLEU[%]	TER[%]	BLEU[%]	TER[%]
1	1.0	26.6	54.2	0.5000	25.9	55.3	30.1	49.6
2	1.3	27.9	51.3	0.4539	27.2	53.0	31.7	47.5
4	1.5	28.7	50.4	0.4387	27.8	52.1	32.1	47.0
8	2.7	28.9	49.9	0.4307	27.8	51.9	32.2	46.9
12	3.3	28.9	50.1	0.4288	27.9	51.6	32.4	46.6
16	4.2	29.0	50.5	0.4277	27.9	51.6	32.4	46.6
32	7.7	28.9	50.8	0.4259	28.0	51.4	32.4	46.6
64	15.1	28.7	52.3	0.4246	28.0	51.3	32.4	47.3
128	29.9	28.3	52.7	0.4233	27.8	51.6	32.3	47.9

by our systems in this task. This allows the hard-alignments to produce a perfect score at most points. The soft-alignments give these points also the highest probability but distributes its probability mass more evenly and receives, therefore, a lower score than the hard-alignment.

To prove this point, we compute the SAER score, also using the hard-alignment that we use to compute the AER score. This gave us a corresponding SAER score that is 10.3% better than its soft equivalent. If we use this score for comparison, the generated alignment outperforms all baseline methods on both evaluation metrics. We obtain an alignment which is superior to the baseline alignments and also to the standard guided alignment approach. In Figure 8.1, we give an example of how the resulting alignments of our different models are. It is visible how the alignment changes from the fuzzy baseline alignment to very accurate alignment points using target foresight.

To verify that the obtained alignments can be used to improve the performance of a neural machine translation system we evaluate the guided alignment training on the IWSLT2013 task. We apply our target foresight alignment model to produce a soft alignment for the Europarl training corpus and use it in guided alignment training. The resulting score of 18.8% BLEU was an improvement of 0.4% BLEU compared to a model trained using the GIZA++ alignment and 2.8% compared to the NMT baseline system. We also observe an improvement of 1.3% AER.

8.5 Analyzing Search

To get the best possible translation according to a model, we need to find the translation which gets the best model score. This search is in neural networks commonly done using a beam search, as described in Section 7.3.4.

8.5.1 Beam Size

The effect of the beam size on different parameters, like the model and BLEU score shown in Table 8.12, is shown on the IWSLT 2013 German→English task. Interestingly, increasing the beam size does not necessarily reduce the BLEU score. We observe the substantial improvement in BLEU by going from beam size 1 to 4. Increasing the beam size to 12 shows further clear improvements while going to a higher number gives only small improvements or in with the

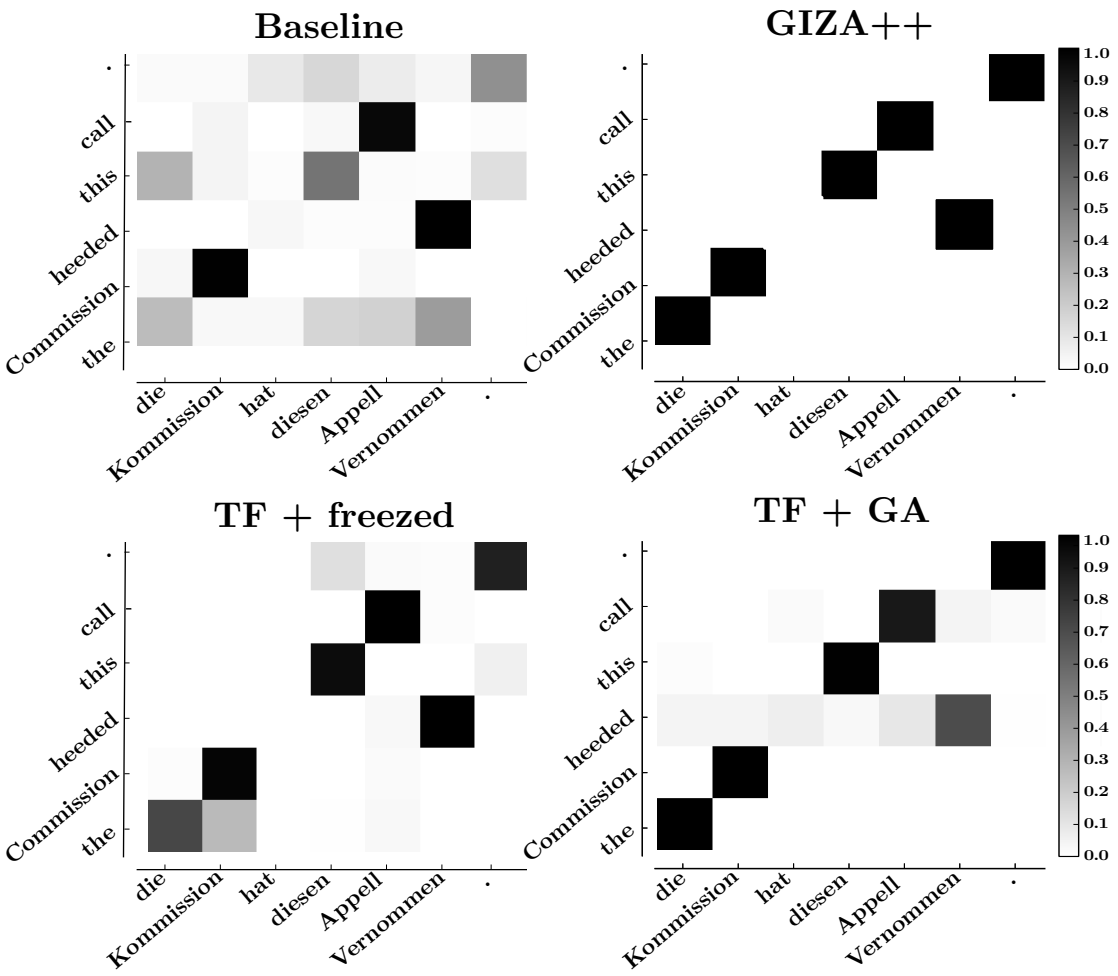


Figure 8.1: Attention weight matrices visualized in heat map form. Generated by the NMT Baseline, GIZA++, target foresight with freezed encoder and decoder parameters (TF + freezed) and target foresight with guided alignment training (TF + GA).

largest settings even worse Bleu scores. Note that our search criterion, the model score, keeps going down with increased model scores as expected. This clearly shows the discrepancy between the model score and the Bleu score. The steeper drop in Bleu on the development set compared to the other sets could be related to the fact that the model was selected using a beam search of size 12.

The search time, shown on CPU using the dev set with 887 sentences, grows as expected approximately linear if we ignore the smaller beam sizes where constant effects are more likely to distort the time.

8.5.2 Comparison of Reference and Hypothesis Cost

To find out if the search is the reason why we miss the reference translations, we compared the score of the translation found using search with a beam size of 12 with the score the reference translation would get. If the reference translation gets a better score than the found hypotheses, it will imply that the model knows which translation is better, but the search does not find it.

The experiment shows that only in six out of 887 translations (0.68%) do the references get a

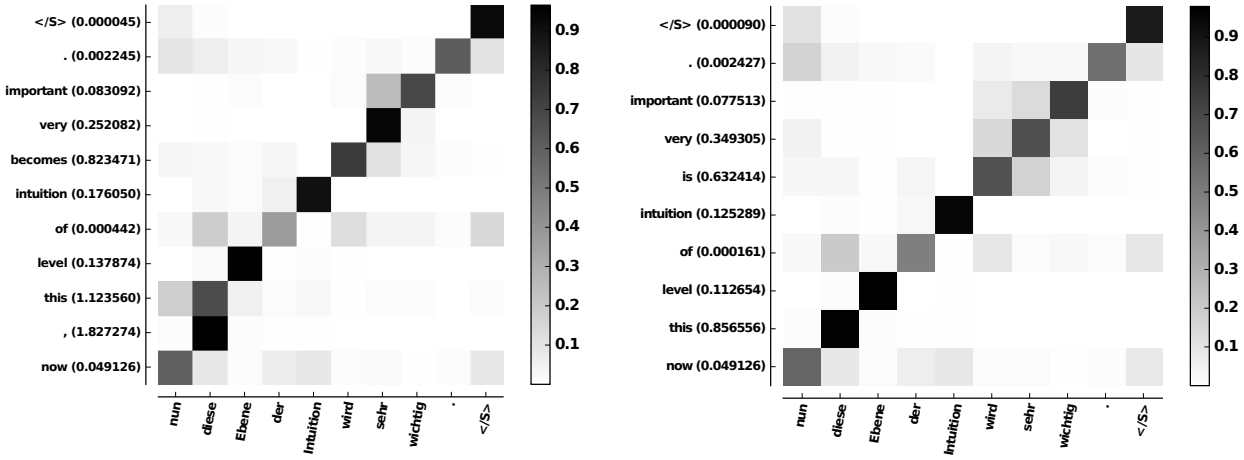


Figure 8.2: The network wants to translate "`dieses`" into "`this`" but is forced to create "`,`" which causes a high cost.

lower score than the hypothesis. If we take a closer look at the attention of these sentences, we find some examples where the model and the reference disagree. Figure 8.2 shows an example where the reference inserts a comma, which the model would like to skip. A possible reason would be since there is no comma on the source side. Another mistake by the model is to translate `wird` to `is` and not `becomes`. In Figure 8.3, we see the problem of a more freely chosen reference translation. The model would like to stay close to the source sentence while the reference splits the translation into two. We see here that a perfectly good translation can get a low model score just because it decides to structure the sentence differently.

8.6 Conclusion

In Section 8.2, we show that the effect of the different methods presented in Section 7.4 is not very large in most cases. The results of Alignment Penalty (Section 8.2.1) and Alignment History (Section 8.2.3) to net present these methods in a favorable way in combination with Adam. Alignment Feedback with Fertility (Section 8.2.2) shows, in most cases, improvement and does not score worse than the baseline in our experiments. The same is true for the Convolutional Alignment Feedback using the setting (k=4, m=2) (Section 8.2.4). While all of these improvements provide less than 1% Bleu, they show the improvements across all three language pairs.

Section 8.3.1 shows that the guided alignment approach gave the largest improvements on the German→English and English→Romanian tasks. While these improvements go up to 1.0% Bleu, this method has the disadvantage of complicating the training setup significantly since it requires to create a word alignment before the regular neural network training can start.

We show in Section 8.4 that attention-based models are capable of generating alignments that improve over the BerkeleyAligner alignments by 1.5% Aer. Using target foresight, we can improve the Aer by 19.1% compared to the baseline attention mechanism and outperform the GIZA++ alignments by 2.0% Aer absolute and 9.5% relative using training with guided alignment. Training the network to produce high-quality alignments proves to be a hard task. The network seems to encode the knowledge of the target word in the attention weights and produces a non-usable alignment if we do not take measures to counteract this.

In Section 8.5, we show that increasing the beam search size does not necessarily lead to better Bleu scores. For the IWSLT 2013 German→English task, we see that the improvement stops

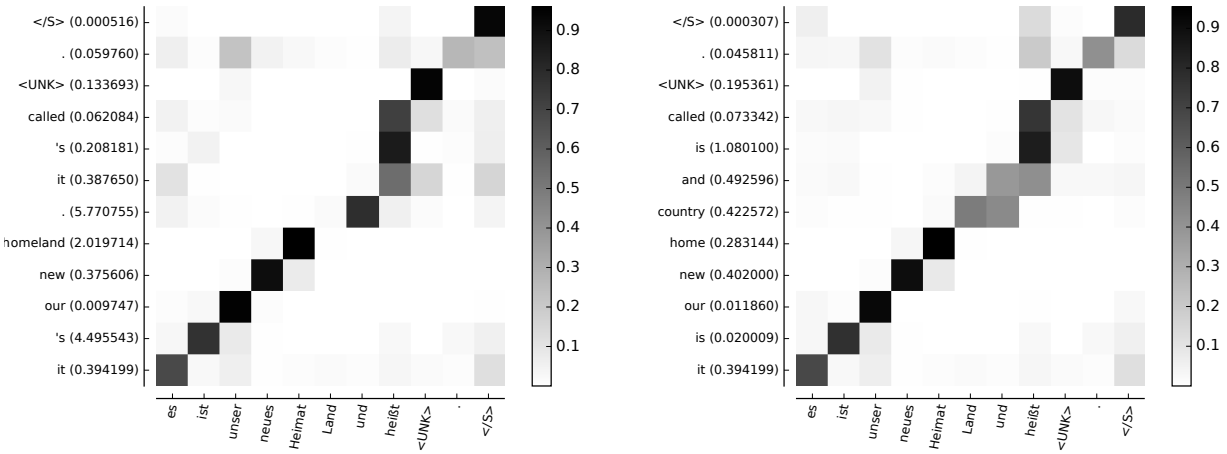


Figure 8.3: The network wants to translate "`und`" but is forced to create "`.`" instead of "`and`", which causes a high cost. The reference translation divides the sentence into two. This is unexpected by the model, which prefers a more literal translation.

between beam-size 12 and 32 and the BLEU score drops for a beam larger than that. We also show that the model only gives in 0.68% of the cases, where it fails to find the reference translation, a better score to the reference translation than to the found translation. This means that we need to improve the modeling rather than search for NMT.

8.7 Contributions

All experiments testing the attention modification in Section 7.4 are done by the author as well as the guided alignment experiments on WMT 2017 German→English, WMT 2016 English→Romanian, and BOLT Chinese→English (Section 8.4). The guided alignment experiments on IWSLT 2013 German→English as well as the target foresight experiments (Section 8.4) are done by Arne Nix as part of his bachelor thesis [Nix 16] which was supervised by the author. The curriculum learning experiments in Section 8.3.2 and the work analyzing the beam search in Section 8.5 was executed by Henrik Rosendahls. The author supervised his bachelor thesis [Rosendahl 16], during which these experiments were done.

9. Experiment Management

Managing experiments for scientific studies or evaluation campaigns can be a cumbersome task that gets quickly out of hand. The experiments that are run should be easily reproducible, allowing to test many possible combinations while avoiding duplicated work to save code, disk space, and most of all time for the user and machines. Each experiment usually has many inputs from many sources and requires many smaller sub-steps.

In this chapter, we describe the task manager, which we developed as part of this thesis.

9.1 Motivation

Building competitive machine learning systems requires the correct execution of a large number of different commands and components.

To build, for example, a machine translation system, we need to pre-process the training data, train a neural network, and evaluate its performance. Each of these steps can consist of a large number of separate steps. To build a phrase-based system even more steps are needed, such as computing a word alignment, extracting the phrases or creating a language model. Running and later replicating all steps by hand is cumbersome and error-prone.

A common approach to reduce these problems is to create ad-hoc scripts for each given task. Although this can be a solution for some parts of the process, it is inflexible if the workflows change as often as required in research. Additionally, errors are easily overlooked when running a large number of scripts in parallel.

We wrote the task manager `Sisyphus` [Peter & Beck$^+$ 18] to ensure that tasks can be easily repeated and managed. Organizing the work this way also allows the user to reconfigure the experiment easily and reuse tested sub-tasks in other workflows. It is designed to handle large and complicated workflows and was already used in practice to handle workflows containing ten thousands of individual tasks.

Finding a good naming scheme for multiple related experiments is also hard since initially good choices often turn out to grow into strange constructs as new experiments are added over time, resulting in names like "ExperimentA-withB-withoutC-D=6-version3". It is often hard to fix naming errors since other experiments might be using parts of it already. We want to map each job to a unique path, which the program can use, and create links bearing descriptive names, which are user friendly. This allows the user to easily rename experiments, if the initial choice turns out to be flawed, without violating any dependencies.

9.2 Basic Assumptions

We structure the workflow by dividing it into 'Jobs'. A job performs a specific function, e.g., evaluating a translation, it usually executes an external script or program.

We built the system on one central assumption: The output of a job depends only on the given input parameters. As an example: Evaluating a translation depends on the hypothesis, the reference, and optionally an evaluation script and some parameters. The result of the computation should not change if the inputs do not change. This property is used to avoid multiple computations of the same job. If randomness is needed, it can still be modeled by using seeds that are given via the job parameters to allow for reliable, reproducible results. If this is not possible, e.g., for asynchronous neural network training, the workflow manager should still work. However, it cannot be guaranteed to reproduce the same result.

This means that the automatic detection of changes in the content of an input file is beyond the scope of this system. If the content of an input file changes, it is necessary to manually run a command to invalidate all jobs that depend on it.

9.3 Design Goals

The design of `Sisyphus` is mainly guided by the problems that we encounter while building statistical machine translation and automatic speech recognition systems. `Sisyphus` aims to address the following problems:

- Separation of the workflow description of an experiment and the place where the experiment is run: This allows the user to store the small workflow description on a safe but expensive file server with backups, while the outputs of an experiment and temporary data are stored on a larger, but less reliable file system (Section 9.6.1 and 9.6.4).
- Reusability of jobs: Once a job is defined, it should be easy to use at a different position within the workflow.
- Minimal requirements on the underlying server structure
- Work definition independent of underlying cluster queuing engine: Moving it to a different queuing engine should be easy, e.g., testing the workflow on a local computer before moving it to a grid engine.
- Avoid redundant computations to save time and disk space by grouping jobs with the same input arguments.
- Start all needed jobs automatically in the correct order and if no blocking dependencies are found, in parallel.
- Automatically checks for errors, report and, if possible, recover them. Errors that occur silently somewhere in the pipeline can cause strange results and are hard to find. (Section 9.5.3)
- Be as general as possible and easy to extend: The whole workflow definition is written in standard conform Python. This allows for much flexibility in defining a workflow and using external tools written for Python e.g., to check and edit the workflow.
- The ability to integrate any external tool that has a command-line interface into a job.

9.4 Related Work

A large variety of heavyweight workflow management systems exist already, e.g. Pegasus [Deelman & Vahi$^+$ 15], Taverna [Wolstencroft & Haines$^+$ 13], Swift [Wozniak & Armstrong$^+$ 13], and

Kepler [Ludäscher & Altintas+ 06]. They can cover a large variety of use-cases [Liew & Atkinson+ 16], but their use is hindered by strict requirements on the user's computing nodes. Moses [Koehn & Hoang+ 07] contains an Experiment Management System (EMS), a complex script written in Perl, for handling the Moses training pipeline.

The toolkit that seems to be most similar to our approach is Ducttape[1], the successor of LonnyBin [Clark & Lavie 10]. It is well designed and covers many useful points. However, we miss a more flexible configuration of the workflow e.g., workflows that adjust to the outputs of finished jobs are not supported. This does not allow to trigger parts of the workflow only if current computations show that they are required.

Ducttape uses branch points to distinguish between different experiment settings. This creates a reasonably intuitive directory structure but does not depend on the real value given to each parameter. If a parameter of a step is changed, it still maps to the same directory. Additionally, long names collapse to a hash value, losing the benefit of intuitively named directories. Interruptions of Ducttape automatically stop all current computations, which makes it problematic to add additional experiments to a running workflow.

9.5 Basic Elements

Every workflow can be modeled as a series of jobs. The output of a job can be either files or parameters (parameters can be seen as a special case of files). The entire workflow can be mapped to a directed acyclic graph where each node is a job, and each edge is a file or parameter. Outputs are either passed on to another job or returned as a result of the workflow.

The user can request the necessary files, and the workflow manager executes all jobs that are needed to compute them. All jobs that are part of the graph, but are not required for the desired output are ignored. This graph structure, as shown in Figure 9.1, is similar to the approach followed by [Clark & Lavie 10].

9.5.1 Jobs

Jobs are the core elements of a workflow and are represented by the nodes in the dependency graph. Every Job has specified inputs and outputs. When an instance of a Job is created, all inputs need to be specified. However, they do not need to exist yet and can be the output of another Job. Once a Job is completed, all of its outputs are assumed to be available for future computations.

After a Job is created, a hash value is computed based on the given input parameters. This hash is used to ensure that only one node is used to represent the same computation. Additionally, it is used as part of the path to the Job directory. The directory where the data produced by the Job is stored. This directory contains log files, status files, the work directory, and the output directory. All commands will be run in this work directory, which is initially empty.

A Job is executed as soon as all inputs are ready. This means all Jobs that compute its inputs are either finished or marked the requested output as done. Before its execution, a Job has the opportunity to request additional inputs. This is necessary to respond to previously specified inputs, e.g., to implement an automatic parameter optimization where we do not know how many iterations are needed. If the Job does not specify additional inputs, it is scheduled for execution in the configured queueing system.

[1] `https://github.com/jhclark/ducttape`

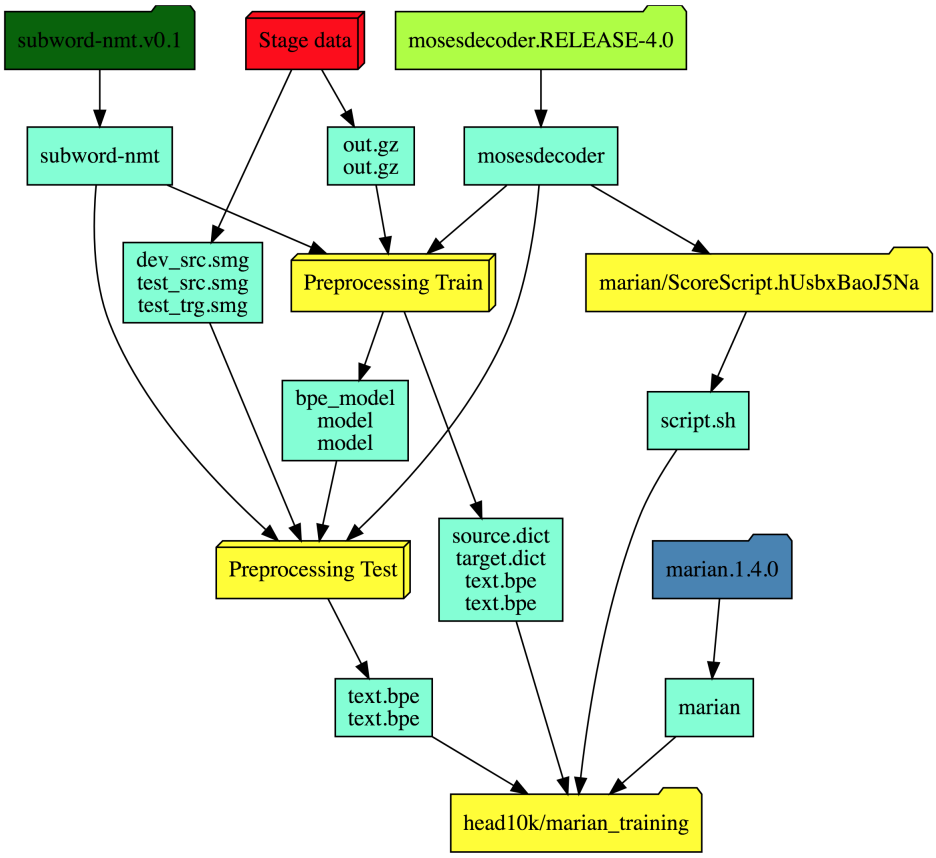


Figure 9.1: Example of a workflow as the `Sisyphus` web interface draws it. Jobs can be grouped to blocks for better visualization as it is done here with the stage data block and the preprocessing blocks. If a Job finishes successfully it is marked dark-green, running Jobs are marked in green-yellow, Jobs that are runnable, but not running yet, are marked blue, Jobs that have to wait for other Jobs to finish are marked yellow, and Jobs that failed are marked red. A block always takes the status of the most problematic Job, e.g., if one Job failed, it is red. If all Jobs are finished, it is green. Files that are shared between Jobs are colored aquamarine.

9.5.2 Tasks

Each Job must have one or more Tasks in which the actual commands that are needed to compute the output are specified. All Tasks that belong to the same Job share the same work directory and are executed in a fixed linear order.

A Task object is the combination of a method of the Job class, a set of requirements, and optionally a set of parameters that will be passed to the method when executed. It is submitted to the grid engine, and once it is scheduled, the given method is executed on the computation node. It is also possible to create arrays of Jobs by providing a list of parameter-sets. These are executed in parallel (to the extent this is supported by the queueing system).

A common approach, for a task running on multiple machines, can be seen in the Job definition shown in Figure 9.2, where the shown Job should apply a given pipeline. The method `tasks` in Line 30 shows how to define which methods should be handled as tasks and with which requirements. We first split the given input into multiple parts with the `split` method in Line 13. Next the command, defined in the `run` method in Line 19, is executed on multiple machines. We indicate that it should be run on multiple machines since the task definition is given a list of

```
1 from sisyphus import * # import all Sisyphus related classes, mainly job and task
2
3 # This is a simple job to demonstrate how to distribute a
4 # pipeline command on to multiple machines
5 class ParallelPipeline(Job):
6     def __init__(self, text, command, parallel_processes=8):
7         self.text = text # Text that will be split and piped though command
8         self.command = command # The actual command
9         # Split into parallel processes
10        self.parallel_processes = parallel_processes
11        self.out = self.output_path('out.gz') # Name of the output path
12
13    def split(self):
14        #Count lines, capture_output gives stdout of command back as string
15        lines = int(self.sh('zcat -f {text} | wc -l', capture_output=True))
16        self.batch_size = (lines // self.parallel_processes) + 1 # compute batch size
17        self.sh('zcat -f {text} | split -d -l {batch_size}') # Split file
18
19    def run(self, pos):  # pos will be given by task
20        # Run the command for each batch
21        self.sh('cat x%02i | {command} > tmp.%02i' % (pos, pos))
22
23    def collect(self):
24        self.sh('cat tmp.* | gzip > {out}')  # collect all outputs
25        # Additional manual sanity check
26        output_lines = int(self.sh('zcat {out} | wc -l', capture_output=True))
27        print("Number of output lines: %i" % output_lines)
28        assert output_lines > 0, "No output created"
29
30    def tasks(self):
31        yield Task('split', rqmt={'cpu': 1, 'mem': 1}) # Run split task first
32        # Continue with the main task and starting a worker
33        # for each element in args list
34        yield Task('run',
35                   rqmt={'cpu': 2, 'mem': 4},
36                   args=list(range(self.parallel_processes)))
37        # Finish with the collect task
38        yield Task('collect',
39                   rqmt={'cpu': 1, 'mem': 1})
```

Figure 9.2: Example of a job containing multiple tasks and running one task on multiple computers

arguments in Line 36. The last task is to collect all computations and write them into the job output file using the `collect` method in Line 23.

9.5.3 Error Handling

We prefer to use strict error checking to avoid that errors are propagated to other jobs. This would often break the workflow at a later point or, even worse, go unnoticed and harm the system performance. This makes it easier to track down the problem that caused the error. By default, a Job switches into an error state if:

- a shell command returns a non-zero value, which is also true for any command inside a pipeline,
- an uninitialized variable is called, or
- the Python code throws an exception, which can be used in combination with assertions.

This means the execution of the failed Job is stopped and the user is notified. If a Task is known to fail spontaneously, it can be set to retry multiple times.

If a Task is interrupted, if it is killed before it is finished executing all commands, it switches into the interrupted state. This usually happens if the Task exceeds the requested requirements. The Task can be marked as resumable if we know that executing the same code multiple times will result in the same output. In this case, the workflow manager automatically tries to determine if the Task got interrupted due to a time or memory limit. It increases the requested requirements automatically and resubmits the Task. Resuming is not performed automatically by default, since some programs behave differently if they find files from previous runs in their work directory. If a Job that is not marked as resumable is interrupted, it stops executing further steps and waits for a manual fix by the user.

9.5.4 Paths and Variables

Jobs are connected by Path and Variable objects, representing the edges in the dependency graph. A Variable is a subclass of the Path object which can store arbitrary pickleable Python objects to be passed between Jobs.

A Job checks all Path objects that are given as inputs and start a Job only once all inputs are available. There are multiple ways for a Path to become available. If it is created as an output of a Job, it is available either once the Job is finished or it is marked as available by the Job earlier. This can be used, for example, if a neural network training creates save points of the current training state, which can be evaluated before the whole training is finished. If a Path object is used to add an input file to the graph, it is marked as available if the file exists and alerts the user otherwise.

9.5.5 Engine

An engine defines how to execute and schedule the given tasks. Currently, supported engines are the Sun Grid Engine (SGE) including its closely related forks, Platform Load Sharing Facility (LSF), and a local engine that is running on the same node as the workflow manager. It is also possible to combine different engines. A typical setup is to have a local engine for small Jobs, e.g., counting the number of lines in a file, and a cluster-based engine for everything else. The choice which engine to use can be given while creating a Task. Currently, all engine implementations require that all nodes have access to a shared file system.

9.6 Directory Structure

If a new set of related experiment is created we set up a directory which contain the following files and directories:

- the `recipe` directory, containing the source code for the Jobs (Section 9.6.1),
- the `config` directory, which defines the Jobs to run and the order of their execution (Section 9.6.2),
- the `settings.py` file, that holds global settings for `Sisyphus`, e.g. which engines are available (Section 9.6.3).

The following directories are automatically created to store the temporary data created by the workflow, the output, and shortcuts to access the jobs.

- the `work` directory, each Job will create a directory here to run its code, store its output, and save log files (Section 9.6.4),

- the `output` directory containing links the finished outputs (Section 9.6.5), and
- the `alias` directory, containing links to running Jobs with given aliases (Section 9.6.5).

9.6.1 Recipe Directory

The recipes are a collection of files that contain the code defining the Jobs that can be executed to run an experiment. Recipe files are regular Python files and can be imported similarly to any other Python module. This allows the users to manage their experiments like a regular Python project, and to create dependencies between Jobs by importing them. What separates the recipe directory from a regular module directory is that is can contain Job definitions. The path to the Job inside the recipe structure will be used to determine the path to the directory where the Job is executed.

9.6.2 Config Directory

The configuration directory is used to create the dependency graph and select which outputs should be computed. Similar to the recipe directory, it contains regular Python files that can be imported. It has to import the needed modules from the recipe directory and create the appropriate Jobs. When starting `Sisyphus`, the user selects which configuration should be loaded to construct the graph.

9.6.3 Settings File

The settings file is used for global parameters. This is the place to, for example, define which engine should be used, if and how the requirements of interrupted Jobs are changed, what the default environment of an executed shell command should look like, various timeouts and many other things.

9.6.4 Work Directory

The work directory stores the realization of the defined graph in the underlying file system. Each Job gets its own directory. Its path is constructed from the recipe module, the Job name, and the hash value of the given inputs. This yields a compromise between structured file names, brevity, and the uniqueness of these names.

Since the work directory contains the output files of all computations, it can grow quite large. The work directory can be completely recreated using the config and recipe directories if each step of the workflow is deterministic. We recommend, therefore, making sure to have a backup of the config and recipe directories, but usually not for the work directory.

9.6.5 Output and Aliases Directory

Outputs that are computed by `Sisyphus` are linked from the work directory to the output directory with the manual given filename. Similarly, it is possible to give Jobs one or more meaningful aliases. While it is possible to access all job computations inside the work directory directly, it is not easy to find it if the hash value is not known. Adding aliases allows having an easy way again to find the right Jobs.

9.7 Available Tools

In this section, we present the most relevant tools to run the workflow and to help with reoccurring tasks.

```
bash-3.2$ ../sis m -r
work does not exist, should I continue? The directory will be created if needed inplace (y/N)y
[2020-01-17 10:41:27,028] INFO: Add target result to jobs (used for more informativ output, disable with SHOW_JOB_TARGETS=False)
[2020-01-17 10:41:27,337] INFO: Experiment directory: /demo/sisyphus/example        Call: ../sis m -r
[2020-01-17 10:41:27,337] INFO: runnable: Job<work/parallel/LineSpliter.vX7RbasMb2vpUGZtTn> <target: result>
[2020-01-17 10:41:27,337] INFO: runnable(1) waiting(1)
[2020-01-17 10:41:27,545] INFO: Submit to queue: work/parallel/LineSpliter.vX7RbasMb2vpUGZtTn run [1]
[2020-01-17 10:41:27,649] INFO: Experiment directory: /demo/sisyphus/example        Call: ../sis m -r
[2020-01-17 10:41:27,649] INFO: queue: Job<work/parallel/LineSpliter.vX7RbasMb2vpUGZtTn> <target: result>
[2020-01-17 10:41:27,649] INFO: queue(1) waiting(1)
[2020-01-17 10:41:29,866] INFO: Experiment directory: /demo/sisyphus/example        Call: ../sis m -r
[2020-01-17 10:41:29,867] INFO: runnable: Job<work/pipeline/Simple.50Shr6wABEk9>
[2020-01-17 10:41:29,867] INFO: runnable: Job<work/pipeline/Simple.AXn8cbrxtANA>
[2020-01-17 10:41:29,867] INFO: runnable: Job<work/pipeline/Simple.DHPOjCROOutr>
[2020-01-17 10:41:29,867] INFO: runnable: Job<work/pipeline/Simple.G7sCGqpuId9m>
[2020-01-17 10:41:29,867] INFO: runnable: Job<work/pipeline/Simple.m6PYks0kvVYu>
[2020-01-17 10:41:29,868] INFO: runnable(5) waiting(41)
[2020-01-17 10:41:29,879] INFO: Submit to queue: work/pipeline/Simple.DHPOjCROOutr run [1]
[2020-01-17 10:41:29,880] INFO: Submit to queue: work/pipeline/Simple.50Shr6wABEk9 run [1]
[2020-01-17 10:41:29,880] INFO: Submit to queue: work/pipeline/Simple.G7sCGqpuId9m run [1]
[2020-01-17 10:41:29,880] INFO: Submit to queue: work/pipeline/Simple.AXn8cbrxtANA run [1]
[2020-01-17 10:41:29,881] INFO: Submit to queue: work/pipeline/Simple.m6PYks0kvVYu run [1]
[2020-01-17 10:41:29,985] INFO: Experiment directory: /demo/sisyphus/example        Call: ../sis m -r
[2020-01-17 10:41:29,985] INFO: queue: Job<work/pipeline/Simple.50Shr6wABEk9>
[2020-01-17 10:41:29,985] INFO: queue: Job<work/pipeline/Simple.AXn8cbrxtANA>
[2020-01-17 10:41:29,985] INFO: queue: Job<work/pipeline/Simple.DHPOjCROOutr>
[2020-01-17 10:41:29,986] INFO: queue: Job<work/pipeline/Simple.G7sCGqpuId9m>
[2020-01-17 10:41:29,986] INFO: queue: Job<work/pipeline/Simple.m6PYks0kvVYu>
[2020-01-17 10:41:29,986] INFO: queue(5) waiting(41)
[2020-01-17 10:41:41,047] INFO: Experiment directory: /demo/sisyphus/example        Call: ../sis m -r
[2020-01-17 10:41:41,048] INFO: queue: Job<work/pipeline/Simple.AXn8cbrxtANA>
[2020-01-17 10:41:41,048] INFO: queue: Job<work/pipeline/Simple.G7sCGqpuId9m>
[2020-01-17 10:41:41,049] INFO: queue: Job<work/pipeline/Simple.m6PYks0kvVYu>
[2020-01-17 10:41:41,049] INFO: running: Job<work/pipeline/Simple.50Shr6wABEk9>
[2020-01-17 10:41:41,049] INFO: running: Job<work/pipeline/Simple.DHPOjCROOutr>
[2020-01-17 10:41:41,049] INFO: queue(3) running(2) waiting(41)
```

Figure 9.3: A screenshot of a running `Sisyphus` manager inside the shell.

9.7.1 Manager

The main function of `Sisyphus` is to manage which Job should he run next and which requirements it has. This is done by starting the manager. If the manager it started, it will either print all relevant events to the system shell as shown in Figure 9.3 with only very few possibilities to interact, or it can be started using the interactive user interface, as shown in Figure 9.4.

9.7.2 Web Server

The web server provides an alternative way to list all Jobs and their current states. It is also possible to show all Jobs in a graph structure, as shown in Figure 9.1. Each Job can be selected to show more detailed information about its status, dependencies, and possible error messages.

9.7.3 Console

It is possible to start an interactive Python console, based on iPython [Pérez & Granger 07], to analyze the graph or test different functions directly. The console also serves to call the team import (Section 9.7.4) and to initialize to automatic clean-up (Section 9.7.5).

9.7.4 Team Import

If multiple people work on overlapping tasks, it saves time and space to avoid rerunning computations that have already been carried out by others. It is possible to automatically check other work directories and import finished Jobs by linking or copying them. This saves one from

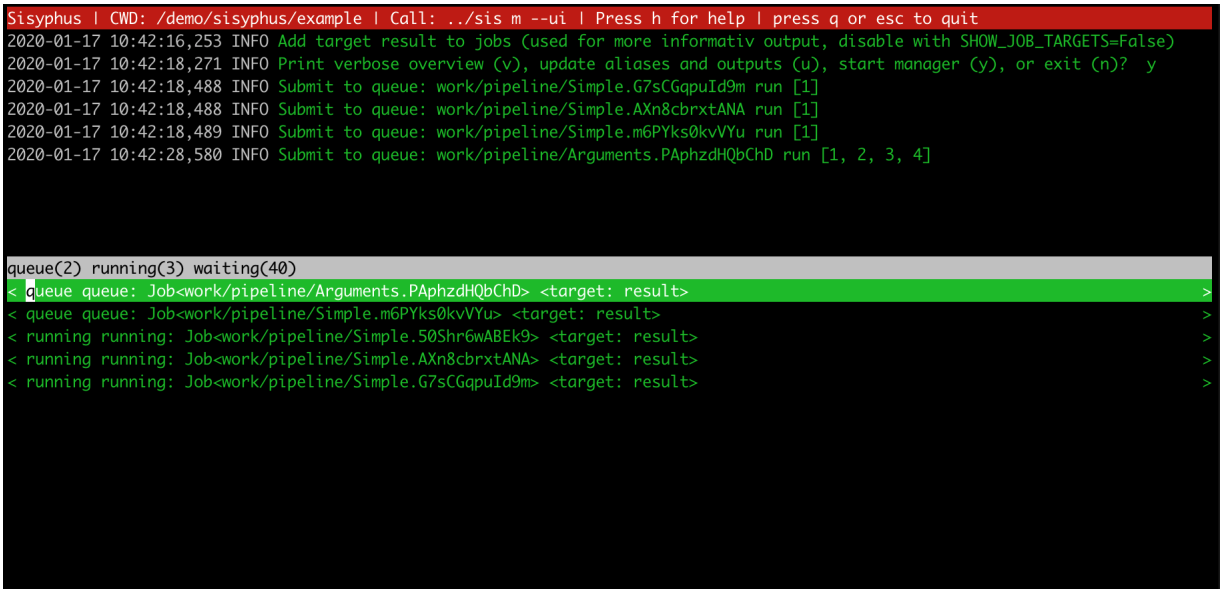


Figure 9.4: A screenshot of the interactive user interface to `Sisyphus` manager based on ncurses.

manually linking finished computations, as it is usually the case when using scripts instead of a workflow manager.

9.7.5 Clean Up

After the experiments are finished it is time to clean up. `Sisyphus` supports a few options to do this, depending on how harsh the clean-up has to be. This is mainly a trade-off between how much space is used on disk vs. how many steps are needed to re-run the experiments.

The least invasive method is to delete the work directory of each Job to remove temporary data created during the execution of the Job and to pack all log files into a tar archive. The workflow manager can be configured to run this clean-up method automatically in the background after a Job has finished successfully.

The second method is to remove lost Job directories from Jobs that are not in the final graph. This usually happens if the workflow changed over time, and some steps had to be re-run due to changed inputs. The now obsolete directories remain on disk until they are removed. An alternative source for lost data in the work directories are Jobs that have been restarted after an error which causes `Sisyphus` to move the old directory aside in case that later debugging is necessary. This step keeps all the data used in the current workflow.

A more invasive option to free space is the clean-up of the current graph by removing Jobs that are not needed for further computations of the workflow anymore. This only keeps Jobs that produce outputs that are marked as targets or Jobs that are still needed to reach unfinished targets. In addition, Jobs can be saved from deletion by defining a score, Jobs with a score higher than the chosen threshold will be kept. These are typically Jobs that are expensive to recompute, e.g., the training of a neural network.

9.8 Usage

`Sisyphus` is used extensively by the machine translation and the automatic speech recognition teams at the RWTH Aachen University. All WMT and IWSLT submissions of the RWTH Aachen University with the author involved since 2015 have been created using `Sisyphus` [Peter

& Toutounchi$^+$ 15b, Peter & Toutounchi$^+$ 15a]. It was used to manage at least some of the experiments in all publications the author participated in recent years and also in other publications [Zeyer & Beck$^+$ 17]. AppTek[2] also uses `Sisyphus` internally.

9.9 Conclusion

The presented overview of the workflow manager `Sisyphus` shows how it helped the author and others to organize their workflow. Features like automatic error detection, efficient usage of computational resources, scalability, easy reproducibility, ability to share work with others have been proven to be extremely helpful during research. The conformity to the Python standard allows using the vast collection of tools available for Python without modification for editing, debugging, and documenting the workflow. It is freely available online[3] under the Mozilla License v2.0 to encourage the adoption by other groups.

9.10 Contributions

The toolkit was designed and written from the ground up by the author of this thesis. Eugen Beck helped with significant design decisions and contributed to many parts. Others contributed additional features or bug fixes, especially Nick Rossenbach and Arne Nix. The toolkit is published in [Peter & Beck$^+$ 18].

[2] `http://www.apptek.com/`
[3] `https://github.com/rwth-i6/sisyphus`

10. Scientific Achievements

This chapter highlights how the scientific goals of this thesis, listed in Chapter 2, are accomplished:

- Chapter 5 shows how to integrate neural networks into a phrase-based system. The results in Chapter 6 show that these neural networks improve the performance of an already heavily optimized phrase-based system by up to 0.9% Bleu. The joint model and the translation model show the most substantial improvements.

  It also presents a method to integrate long-distance dependencies into a feed-forward neural network by adding an input for bag-of-words. Adding this input gives an additional improvement of up to 0.5% Bleu. Most of the time, the improvement did not reach 0.5% Bleu in our experiment, but we observe a consistent improvement over the baseline. Tests show that this improvement is not replicated by merely enlarging the input window size. A comparison with a recurrent LSTM translation model shows that the bag-of-words as input performs on the same level. If we use the neural translation model in decoding instead of rescoring, we see an improvement in the performance of approximately 0.3% Bleu.

- In Chapter 7 we present our work on the encoder-decoder model with attention and show the experimental results in the following Chapter 8. These experiments show that adding direct knowledge of previous attention to the current attention computation improves the performance of the model if the right method is used. Using alignment feedback (Section 7.4.2) or the convolutional alignment feedback (Section 7.4.4), we saw improvements over the baseline of up to 0.5% Bleu. On the other hand, the alignment penalty (Section 7.4.1) and the alignment history (Section 7.4.3) on the other hand did not improve over the baseline model.

  In Section 7.5.1, we show how pre-computed word alignments can be used to transfer the knowledge of an external word alignment to the neural network during training. We see improvements of up to 1.0% Bleu (Section 8.3.1) in model performance. We also observed an improvement of 6.4% Aer compared to the baseline system. The experiments with curriculum training in Section 8.3.2 so far show no improvement. More settings need to be tested to find a working combination.

  In Section 7.6 we propose a novel method of how neural network translation systems can be modified to compute word alignments. The experiments in Section 8.4 showed that we are able to improve the Aer over state of the art by 2.0% using this method. Compared to the baseline NMT system, we observed an improvement of 19.1% Aer.

  Analyzing the beam search of neural models (Section 8.5) showed two interesting properties. The first property is that a larger beam tends to hurt the model performance after a certainty threshold. The Bleu score decreases even though the cost of the final translation decreases as well. An additional result of our analysis is that the reference gets a worse score than

the used translation most of the time. Only in less than 1% of the cases does the reference get a better score. We conclude from this that improving the model is more important than the search to improve the overall performance of the system.

- The Sisyphus toolkit presented in Chapter 9 was built as part of this thesis. It provides a way of how to manage and reproduce a potentially large number of complicated experimental setups. We made heavy use of it during research and evaluations for this thesis.

  This toolkit is freely available online under the Mozilla License v2.0. A number of students at the department are using it nowadays. The toolkit is also used outside of the RWTH for example at AppTek[1].

- The experiments in Chapter 6 and Chapter 8 show the performance of the developed methods on publicly available large scale tasks.

  These methods were additionally also tested by participating at several large scale evaluation campaigns. All submissions by the author to these campaigns are built with Sisyphus. Most of these submissions incorporated thousands of tasks and subtasks, demonstrating the scalability of the workflow manager. The campaigns are:

  - In the WMT 2015 German→English task [Bojar & Chatterjee$^+$ 15] we used multiple neural networks in combination with a phrase-based model as described in Section 5.3 [Peter & Toutounchi$^+$ 15b], including models which used bag-of-words as input. We reached a close third place out of eight systems in BLEU. If unrestricted online systems are included, we reached fourth place of 12 systems.
  - We achieved the first place out of three submissions in the IWSLT 2015 German→English machine translation task [Cettolo & Niehues$^+$ 15]. The system was based again based on the phrase-based approach supported by neural networks [Peter & Toutounchi$^+$ 15a].
  - The author submitted two systems to the WMT 2016 evaluation campaign [Bojar & Chatterjee$^+$ 16] for the English→Romanian task. The system representing a cooperation effort with QT21/HimL [Peter & Alkhouli$^+$ 16b] reached first place in BLEU out of 12 systems (including two online systems). It was a system combination [Leusch & Freitag$^+$ 11] of 12 systems built using Sisyphus.

    The system submitted separately by the RWTH reached third place [Peter & Alkhouli$^+$ 16a] according to BLEU and in the second group according to the human evaluation. It makes use of three separate systems combined in a system combination. A neural machine translation system, as described in Section 7.3, a phrase-based, and a hierarchical phrase-based system. The last two are supported by neural networks.
  - In the IWSLT 2016 German→English [Cettolo & Niehues$^+$ 16] we reached first place on all three test sets. Each task had between three and five participants. We used a system combination using phrase-based systems and a large variety of neural machine translation systems [Peter & Guta$^+$ 16]. Multiple variants of the attention modifications described in Section 8.2 and the guided alignment training as described in Section 7.5.1 were included.
  - We submitted one system to the WMT 2017 German→English task [Bojar & Chatterjee$^+$ 17] and a system in the other direction. In the human evaluation, we reached the first cluster with five other systems for German→English out of 11 systems in total. For the English→German task, we reached the second cluster with eight other systems out of

[1] `http://www.apptek.com`

a total of 16 systems (the first cluster contained only one system). The submitted systems are ensembles out of multiple neural machine translation models [Peter & Guta$^+$ 17]. Some of these models incorporated attention modifications presented in Section 7.4.

These submissions demonstrate that the methods presented in this thesis work on large scale evaluation tasks. Bridging the gap, starting with the phrase-based system by adding feed-forward neural networks, shows clear improvements. The same is true if we start with neural machine translation systems. We see that introducing the alignment concepts, known from the statistic machine translation systems, to them improved their performance, either by changing the model structure or by copying an externally created word alignment. Both the phrase-based approach and the neural machine translation approach can profit from the other.

11. Individual Contributions

In this chapter, we list in the contribution of the author of this thesis to the methods and experiments described in this thesis as well as his contribution to the published papers.

11.1 Feed-forward Neural Networks

The feed-forward neural network implementation used in Chapter 5 and Chapter 6 was written from ground up by the author using the Theano framework [Bergstra & Breuleux$^+$ 10].

The addition of bag-of-words models was implemented and tested by Weiyue Wang as part of his master thesis, which was supervised by the author. This work was also published in:

- *Exponentially Decaying Bag-of-Words Input Features for Feed-Forward Neural Network in Statistical Machine Translation* [Peter & Wang$^+$ 16]

Weiyue Wang executed in the same context the experiments on IWSLT 2013 German→English, which are shown in Chapter 6. The author of this thesis executed all other experiments in that chapter. The same neural network implementation was also utilized for:

- *A Comparison between Count and Neural Network Models Based on Joint Translation and Reordering Sequences* [Guta & Alkhouli$^+$ 15]

All experiments related to feed-forward neural networks in that work were conducted by the author of this thesis.

11.2 Neural Machine Translation

The author set up and adapted the Blocks framework [Van Merriënboer & Bahdanau$^+$ 15] to be usable for the needs at the department. This codebase was later used for research and evaluations.

The implementation of the attention modifications, alignment penalty (Section 7.4.1), alignment feedback (Section 7.4.2), fertility (Section 7.4.2), and alignment history (Section 7.4.3), was done in the scope of Nick Rossenbach's bachelor thesis [Rossenbach 16] which was closely supervised by the author.

The author also supervised the bachelor thesis of Arne Nix [Nix 16], who implemented in that context the convolutional alignment feedback (Section 7.4.4) as additional attention modification. All experiments testing the attention modification in Section 7.4 are done by the author.

The guided alignment training idea was proposed by Evgeny Matusov and Wenhu Chen and refined in personal conversations with the author, who is a co-author of:

- *Guided Alignment Training for Topic-Aware Neural Machine Translation* [Chen & Matusov$^+$ 16]

The here described implementation was done by Arne Nix (Section 7.5.1) as well as the experiments on IWSLT 2013 German→English in Section 8.3.1. The other experiments on guided alignment were done by the author of this thesis.

The author had the idea for target foresight and setup the experiment design shown in Section 7.6 and Section 8.4. Arne Nix implemented it and tested it also as part of his bachelor thesis. The author published this work in:

- *Generating Alignments Using Target Foresight in Attention-Based Neural Machine Translation* [Peter & Nix$^+$ 17]

The implementation of the curriculum learning, as described in Section 7.5.2, and the experiments in Section 8.3.2 were done by Henrik Rosendahl as part of his bachelor thesis [Rosendahl 16], which was supervised by the author. Similar for the beam search experiments in Section 8.5.

The same framework for neural machine translation was used to create comparison experiments with attention-based systems for the publication by Tamer Alkhouli:

- *Alignment-Based Neural Machine Translation* [Alkhouli & Bretschner$^+$ 16]

Parnia Bahar used the framework as well and was supported with setting up and designing the experiments by the author of this thesis for this work:

- *Empirical Investigation of Optimization Algorithms in Neural Machine Translation* [Bahar & Alkhouli$^+$ 17]

11.3 Toolkits

The Sisyphus toolkit was designed and written from the ground up by the author of this thesis. Eugen Beck helped with significant design decisions and contributed to many parts. Various users provided smaller contributions, especially Nick Rossenbach and Arne Nix. The toolkit was published in:

- *Sisyphus, a Workflow Manager Designed for Machine Translation and Automatic Speech Recognition* [Peter & Beck$^+$ 18]

The author also contributed to the Jane 2 toolkit, mainly by adding the dependency language model and alignment extraction as well as bugfixes:

- *Jane 2: Open Source Phrase-based and Hierarchical Statistical Machine Translation* [Wuebker & Huck$^+$ 12]
- *Hierarchical Phrase-Based Translation with Jane 2* [Huck & Peter$^+$ 12]

The author also supported the development of the CharacTER toolkit. The implementation and most of the experiments are done by Weiyue Wang. The work is published in:

- *CharacTER: Translation Edit Rate on Character Level* [Wang & Peter$^+$ 16]

11.4 International Evaluation Campaigns

The author participated in a number of evaluation campaigns. For the WMT 2013 submission, he contributed major parts to the English→French systems:

- *The RWTH Aachen Machine Translation System for WMT 2013* [Peitz & Mansour$^+$ 13]

The neural language models used in the IWSLT 2013 submission, namely for English→French and English→German, were trained and applied by the author:

- *The RWTH Aachen Machine Translation Systems for IWSLT 2013* [Wuebker & Peitz$^+$ 13a]

The author of this thesis organized the submission of the department for the following submissions. He was mainly responsible for all German→English submissions as well as the English→Romanian submission. Methods like the bag-of-word models (Section 5.4) or attention based model with modified attention (Section 7.4) were used in all these submissions. Sisyphus (Chapter 9) was used to manage these submissions:

- The RWTH Aachen German-English Machine Translation System for WMT 2015 [Peter & Toutounchi$^+$ 15b]
- The RWTH Aachen German to English MT System for IWSLT 2015 [Peter & Toutounchi$^+$ 15a]
- The RWTH Aachen University English-Romanian Machine Translation System for WMT 2016 [Peter & Alkhouli$^+$ 16a]
- The RWTH Aachen Machine Translation System for IWSLT 2016 [Peter & Guta$^+$ 16]
- The RWTH Aachen University English-German and German-English Machine Translation System for WMT 2017 [Peter & Guta$^+$ 17]

The author submitted a further two submissions in cooperation with other teams from QT21 and HimL. For these submissions, the author collected the translations from all team members and created the system combination. For the submission to WMT 2016, the author additionally provided a trainied system.

- The QT21/HimL Combined Machine Translation System [Peter & Alkhouli$^+$ 16b]
- The QT21 Combined Machine Translation System for English to Latvian [Peter & Ney$^+$ 17]

11.5 Other Work

The author also contributed to work which was not directly related to his thesis. For these reordering models proposed by Minwei Feng, trained and applied the author of this thesis, these were recurrent neural network models with LSTM layers.

- *Reordering: Advancements in Reordering Models for Statistical Machine Translation* [Feng & Peter$^+$ 13]

- *Sequence Labeling-based Reordering Model for Phrase-based SMT* [Feng & Peter$^+$ 12]

For the conditional random field experiments, the author of this thesis contributed the n-best list used for rescoring.

- *(Hidden) Conditional Random Fields Using Intermediate Classes for Statistical Machine Translation* [Lehnen & Peter$^+$ 13]

The author also supported this work with a slightly modified version of the NPLM toolkit [Vaswani & Zhao$^+$ 13] and helped with the experiment design.

- *Local System Voting Feature for Machine Translation System Combination* [Freitag & Peter$^+$ 15]

A. Overview of the Corpora

A.1 Conference on Machine Translation (WMT)

The Conference on Machine Translation organizes an annual evaluation campaign. We use the news translation task for which they provide training and test data for the participating language pairs. The abbreviation *WMT* comes from the fact that the conference was formerly known as *Workshop on Statistical Machine Translation*.

A.1.1 WMT 2017 German→English

The corpus statistics for the *EMNLP 2017 Second Conference On Machine Translation (WMT17)*[1] German→English task are shown in Table A.1.

Table A.1: Corpus statistics for WMT 2017 German→English task. The 30K Vocabulary OOVs lists the number of OOVs if the vocabulary is build using the 30K most frequent words seen in the training data.

`train`:	Sentences	3.57M	
	Running Words	85.06M	86.43M
	Vocabulary	674.4K	590.3K
	Singletons	271.9K	319.8K
`newstest2015 (dev)`:	Sentences	2169	
	Running Words	48.3K	47.0K
	OOVs	1016 (2.1%)	1356 (2.9%)
	30K Vocabulary OOVs	6429 (13.3%)	3153 (6.7%)
`newstest2016`:	Sentences	2999	
	Running Words	69.5K	64.7K
	OOVs	1660 (2.4%)	2098 (3.2%)
	30K Vocabulary OOVs	9922 (14.3%)	4626 (7.2%)
`newstest2017`:	Sentences	3004	
	Running Words	67.2K	65.0K
	OOVs	1590 (2.4%)	1918 (3.0%)
	30K Vocabulary OOVs	9800 (14.6%)	4475 (6.9%)

[1] `http://www.statmt.org/wmt17/`

A.1.2 WMT 2016 English → Romanian

The corpus statistics for the *ACL 2016 First Conference On Machine Translation (WMT16)*[2] English→Romanian task are shown in Table A.2. The organizers provided the newsdev2016 development set together with the training data and the newstest2016 sets after the evaluation. The newsdev2016 was split into two equal sets to allow for having a dev and a test set before the newstest2016 test set was made available.

Table A.2: Corpus statistics for WMT 2016 English→Romanian task. The 30K Vocabulary OOVs lists the number of OOVs if the vocabulary is build using the 30K most frequent words seen in the training data.

`train`:	Sentences	603.6K	
	Running Words	15.46M	15.76M
	Vocabulary	92.3K	128.4K
	Singletons	36.2K	47.4K
`newsdev2016/1 (dev)`:	Sentences	1000	
	Running Words	25.6K	26.7K
	OOVs	1802 (7.0%)	1562 (5.9%)
	OOVs limited	2464 (9.6%)	3045 (11.4%)
`newsdev2016/2`:	Sentences	999	
	Running Words	26.3K	25.6K
	OOVs	1792 (6.8%)	1607 (6.3%)
	OOVs limited	2352 (8.9%)	3304 (12.9%)
`newstest2016`:	Sentences	1999	
	Running Words	50.1K	49.5K
	OOVs	3395 (6.8%)	3053 (6.2%)
	OOVs limited	4459 (8.9%)	6354 (12.8%)

[2] `http://www.statmt.org/wmt16/`

A.2 Broad Operational Language Translation (BOLT)

The *DARPA Broad Operational Language Translation* (BOLT) Program[3] was launched in October 2011 to create new techniques for automated translation that can be applied to the informal genres of text and speech common in online and in-person communication. The training and test data were provided to all organizations that participated in the program.

A.2.1 BOLT Chinese→English

The corpus statistics for the BOLT Chinese→English task are shown in Table A.3.

Table A.3: Corpus statistics for BOLT Chinese→English task. The 30K Vocabulary OOVs lists the number of OOVs if the vocabulary is build using the 30K most frequent words seen in the training data.

`train`:	Sentences	4.08M	
	Running Words	78.30M	85.88M
	Vocabulary	383.5K	817.1K
	Singletons	83.6K	571.0K
`dev12-tune (dev)`:	Sentences	1845	
	Running Words	38.3K	46.7K
	OOVs	1018 (2.7%)	921 (2.0%)
	OOVs limited	2591 (6.8%)	1496 (3.2%)
`dev12-dev`:	Sentences	1844	
	Running Words	38.6K	48.4K
	OOVs	891 (2.3%)	754 (1.6%)
	OOVs limited	2340 (6.1%)	1256 (2.6%)
`P1R6-dev`:	Sentences	1124	
	Running Words	24.0K	28.8K
	OOVs	604 (2.5%)	595 (2.1%)
	OOVs limited	1530 (6.4%)	878 (3.0%)

[3] `https://www.darpa.mil/program/broad-operational-language-translation`

A.3 International Workshop on Spoken Language Translation (IWSLT)

The *International Workshop on Spoken Language Translation* (IWSLT) is an annual evaluation campaign which focuses on spoken language recognition and translation. The test set is often extracted from TED tasks[4] as well as the provided indomain data.

A.3.1 IWSLT 2013 German → English

The corpus statistics for the *IWSLT 2013, International Workshop on Spoken Language Translation*[5] German→English task are shown in Table A.4.

Table A.4: Corpus statistics for IWSLT 2013 German→English task. The 30K Vocabulary OOVs lists the number of OOVs if the vocabulary is build using the 30K most frequent words seen in the training data.

		German	English
`train (full data):`	Sentences	4.32M	
	Running Words	108.46M	108.57M
	Vocabulary	836.2K	792.3K
	Singletons	425.3K	432.8K
`train (in-domain)`	Sentences	137.9K	
	Running Words	2.63M	2.70M
	Vocabulary	75.4K	50.2K
`dev`	Sentences	887	
	Running Words	20K	20.1K
	OOVs with full vocabulary	468 (2.3%)	197 (0.9%)
	OOVs with 30K shortlist	1346 (6.7%)	656 (3.3%)
`test`	Sentences	1565	
	Running Words	31.6K	32.6K
	OOVs with full vocabulary	677 (2.1%)	1377 (4.4%)
	OOVs with 30K shortlist	1811 (5.7%)	2000 (6.4%)
`eval11`	Sentences	1436	
	Running Words	27.2K	27.6K
	OOVs with full vocabulary	449(1.6%)	1110(4.1%)
	OOVs with 30K shortlist	1526 (5.6%)	1716 (6.5%)

[4] `http://www.ted.com/talks`
[5] `http://workshop2013.iwslt.org/`

List of Figures

List of Tables

A. Bibliography

[A. Clark & G. Farley 55] W. A. Clark, B. G. Farley: Generalization of pattern recognition in a self-organizing system. Vol. , pp. 86–91, 01 1955.

[Alkhouli & Bretschner$^+$ 16] T. Alkhouli, G. Bretschner, J.T. Peter, M. Hethnawi, A. Guta, H. Ney: Alignment-Based Neural Machine Translation. In *ACL 2016 First Conference on Machine Translation*, pp. 54–65, Berlin, Germany, Aug. 2016.

[Alkhouli & Bretschner$^+$ 18] T. Alkhouli, G. Bretschner, H. Ney: On The Alignment Problem in Multi-Head Attention-Based Neural Machine Translation. In *EMNLP 2018 Third Conference on Machine Translation*, pp. 177–185, Brussels, Belgium, Oct. 2018.

[Alkhouli & Ney 17] T. Alkhouli, H. Ney: Biasing Attention-Based Recurrent Neural Networks Using External Alignment Information. In *EMNLP 2017 Second Conference on Machine Translation*, pp. 108–117, Copenhagen, Denmark, Sept. 2017.

[Alkhouli & Rietig$^+$ 15] T. Alkhouli, F. Rietig, H. Ney: Investigations on Phrase-based Decoding with Recurrent Neural Network Language and Translation Models. In *EMNLP 2015 Tenth Workshop on Statistical Machine Translation*, pp. 294–303, Lisbon, Portugal, Sept. 2015.

[Auli & Galley$^+$ 13] M. Auli, M. Galley, C. Quirk, G. Zweig: Joint Language and Translation Modeling with Recurrent Neural Networks. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1044–1054, Seattle, Washington, USA, Oct. 2013. Association for Computational Linguistics.

[Bahar & Alkhouli$^+$ 17] P. Bahar, T. Alkhouli, J.T. Peter, C.J.S. Brix, H. Ney: Empirical Investigation of Optimization Algorithms in Neural Machine Translation. In *Conference of the European Association for Machine Translation*, pp. 13–26, Prague, Czech Republic, June 2017.

[Bahdanau & Cho$^+$ 15] D. Bahdanau, K. Cho, Y. Bengio: Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of the 3rd International Conference on Learning Representations*, San Diego, CA, USA, May 2015.

[Banerjee & Lavie 05] S. Banerjee, A. Lavie: METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pp. 65–72, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[Bar-Hillel 51] Y. Bar-Hillel: The Present State of Research on Mechanical Translation. *American Documentation*, Vol. 2, pp. 229–237, 1951.

[Bengio & Ducharme$^+$ 03] Y. Bengio, R. Ducharme, P. Vincent, C. Janvin: A Neural Probabilistic Language Model. *J. Mach. Learn. Res.*, Vol. 3, pp. 1137–1155, March 2003.

[Bengio & Lamblin+ 07] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle et al.: Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, Vol. 19, pp. 153, 2007.

[Bengio & Louradour+ 09] Y. Bengio, J. Louradour, R. Collobert, J. Weston: Curriculum Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, 41–48, New York, NY, USA, 2009. Association for Computing Machinery.

[Bengio & Simard+ 94] Y. Bengio, P. Simard, P. Frasconi: Learning Long-term Dependencies with Gradient Descent is Difficult. *Trans. Neur. Netw.*, Vol. 5, No. 2, pp. 157–166, March 1994.

[Bergstra & Breuleux+ 10] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, Y. Bengio: Theano: A CPU and GPU math compiler in Python. In *Proc. 9th Python in Science Conf*, pp. 1–7, 2010.

[Bishop 95] C.M. Bishop: *Neural networks for pattern recognition*. Oxford university press, January 1995.

[Bojar & Chatterjee+ 15] O. Bojar, R. Chatterjee, C. Federmann, B. Haddow, M. Huck, C. Hokamp, P. Koehn, V. Logacheva, C. Monz, M. Negri, M. Post, C. Scarton, L. Specia, M. Turchi: Findings of the 2015 Workshop on Statistical Machine Translation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pp. 1–46, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics.

[Bojar & Chatterjee+ 16] O. Bojar, R. Chatterjee, C. Federmann, Y. Graham, B. Haddow, M. Huck, A. Jimeno Yepes, P. Koehn, V. Logacheva, C. Monz, M. Negri, A. Névéol, M. Neves, M. Popel, M. Post, R. Rubino, C. Scarton, L. Specia, M. Turchi, K. Verspoor, M. Zampieri: Findings of the 2016 Conference on Machine Translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pp. 131–198, Berlin, Germany, Aug. 2016. Association for Computational Linguistics.

[Bojar & Chatterjee+ 17] O. Bojar, R. Chatterjee, C. Federmann, Y. Graham, B. Haddow, S. Huang, M. Huck, P. Koehn, Q. Liu, V. Logacheva, C. Monz, M. Negri, M. Post, R. Rubino, L. Specia, M. Turchi: Findings of the 2017 Conference on Machine Translation (WMT17). In *Proceedings of the Second Conference on Machine Translation*, pp. 169–214. Association for Computational Linguistics, 2017.

[Brown & Cocke+ 88] P. Brown, J. Cocke, S.D. Pietra, V.D. Pietra, F. Jelinek, R. Mercer, P. Roossin: A statistical approach to language translation. In *Proceedings of the 12th conference on Computational linguistics - Volume 1*, COLING '88, pp. 71–76, Stroudsburg, PA, USA, 1988. Association for Computational Linguistics.

[Brown & Cocke+ 90] P.F. Brown, J. Cocke, S.A.D. Pietra, V.J.D. Pietra, F. Jelinek, J.D. Lafferty, R.L. Mercer, P.S. Roossin: A statistical approach to machine translation. *ComputationaL linguistics*, Vol. 16, No. 2, pp. 79–85, 1990.

[Brown & Della Pietra+ 93] P.F. Brown, S.A. Della Pietra, V.J. Della Pietra, R.L. Mercer: The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, Vol. 19, No. 2, pp. 263–311, June 1993.

[Cettolo & Niehues+ 13] M. Cettolo, J. Niehues, S. Stüker, L. Bentivogli, M. Federico: Report on the 10 th IWSLT Evaluation Campaign. In *International Workshop on Spoken Language Translation*, Heidelberg, Germany, Dec. 2013.

[Cettolo & Niehues$^+$ 15] M. Cettolo, J. Niehues, S. Stüker, L. Bentivogli, R. Cattoni, M. Federico: The IWSLT 2015 Evaluation Campaign. In *International Workshop on Spoken Language Translation*, Da Nang, Vietnam, Dec. 2015.

[Cettolo & Niehues$^+$ 16] M. Cettolo, J. Niehues, S. Stüker, L. Bentivogli, R. Cattoni, M. Federico: The IWSLT 2016 Evaluation Campaign. In *International Workshop on Spoken Language Translation*, Seattle, Washington, USA, Dec. 2016.

[Chen & Firat$^+$ 18] M.X. Chen, O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, N. Parmar, M. Schuster, Z. Chen, Y. Wu, M. Hughes: The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation, 2018.

[Chen & Goodman 96] S.F. Chen, J. Goodman: An Empirical Study of Smoothing Techniques for Language Modeling. In *34th Annual Meeting of the Association for Computational Linguistics*, pp. 310–318, Santa Cruz, California, USA, June 1996. Association for Computational Linguistics.

[Chen & Kuhn$^+$ 11] B. Chen, R. Kuhn, G. Foster, H. Johnson: Unpacking and Transforming Feature Functions: New Ways to Smooth Phrase Tables. In *Proceedings of MT Summit XIII*, pp. 269–275, Xiamen, China, September 2011.

[Chen & Matusov$^+$ 16] W. Chen, E. Matusov, S. Khadivi, J.T. Peter: Guided Alignment Training for Topic-Aware Neural Machine Translation. Austin, Texas, 2016. Association for Machine Translation in the Americas.

[Cho & van Merriënboer$^+$ 14a] K. Cho, B. van Merriënboer, D. Bahdanau, Y. Bengio: On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pp. 103–111, Doha, Qatar, Oct. 2014. Association for Computational Linguistics.

[Cho & van Merrienboer$^+$ 14b] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio: Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734. Association for Computational Linguistics, October 2014.

[Chorowski & Bahdanau$^+$ 15] J.K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, Y. Bengio: Attention-Based Models for Speech Recognition. In C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pp. 577–585. Curran Associates, Inc., 2015.

[Chung & Gulcehre$^+$ 14] J. Chung, C. Gulcehre, K. Cho, Y. Bengio: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, 2014. cite arxiv:1412.3555Comment: Presented in NIPS 2014 Deep Learning and Representation Learning Workshop.

[Clark & Lavie 10] J.H. Clark, A. Lavie: LoonyBin: Keeping Language Technologists Sane through Automated Management of Experimental (Hyper)Workflows. In *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2010, 17-23 May 2010, Valletta, Malta*, 2010.

[Clarkson & Robinson 97] P. Clarkson, A. Robinson: Language Model Adaptation Using Mixtures and an Exponentially Decaying Cache. In *Proceedings of the 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 799–802, Washington, DC, USA, April 1997.

[Cohn & Hoang+ 16] T. Cohn, C.D.V. Hoang, E. Vymolova, K. Yao, C. Dyer, G. Haffari: Incorporating structural alignment biases into an attentional neural translation model. *arXiv preprint arXiv:1601.01085*, Vol. , 2016.

[Cybenko 89] G. Cybenko: Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, Vol. 2, No. 4, pp. 303–314, 1989.

[Deelman & Vahi+ 15] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P.J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, K. Wenger: Pegasus, a Workflow Management System for Science Automation. *Future Gener. Comput. Syst.*, Vol. 46, No. C, pp. 17–35, May 2015.

[Devlin & Zbib+ 14] J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. Schwartz, J. Makhoul: Fast and Robust Neural Network Joint Models for Statistical Machine Translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pp. 1370–1380, Baltimore, MD, USA, June 2014.

[Doddington 02] G. Doddington: Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*, HLT '02, pp. 138–145, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[Doetsch & Golik+ 17] P. Doetsch, P. Golik, H. Ney: A comprehensive study of batch construction strategies for recurrent neural networks in MXNet. *CoRR*, Vol. abs/1705.02414, 2017.

[Duchi & Hazan+ 11] J. Duchi, E. Hazan, Y. Singer: Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, Vol. 12, No. Jul, pp. 2121–2159, 2011.

[Dyer & Chahuneau+ 13] C. Dyer, V. Chahuneau, N.A. Smith: A simple, fast, and effective reparametrization of IBM model 2. In *Proceedings of the NAACL 7th Workshop on Syntax, Semantics and Structure in Statistical Translation*, Atlanta, Georgia, USA, June 2013.

[European Commission 12] European Commission: Special Eurobarometer 386, Europeans and their Languages. `https://ec.europa.eu/commfrontoffice/publicopinion/archives/ebs/ebs_386_en.pdf`, 2012. [Online; accessed 20-January-2020].

[Feng & Liu+ 16] S. Feng, S. Liu, M. Li, M. Zhou: Implicit Distortion and Fertility Models for Attention-based Encoder-Decoder NMT Model. *arXiv preprint arXiv:1601.03317*, Vol. , 2016.

[Feng & Peter+ 12] M. Feng, J.T. Peter, H. Ney: Sequence Labeling-based Reordering Model for Phrase-based SMT. In *International Workshop on Spoken Language Translation*, pp. 260–267, Hong Kong, Dec. 2012.

[Feng & Peter+ 13] M. Feng, J.T. Peter, H. Ney: Advancements in Reordering Models for Statistical Machine Translation. In *Annual Meeting of the Assoc. for Computational Linguistics*, pp. 322–332, Sofia, Bulgaria, Aug. 2013.

[Freitag & Al-Onaizan 17] M. Freitag, Y. Al-Onaizan: Beam Search Strategies for Neural Machine Translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pp. 56–60, Vancouver, Aug. 2017. Association for Computational Linguistics.

[Freitag & Peter+ 15] M. Freitag, J.T. Peter, S. Peitz, M. Feng, H. Ney: Local System Voting Feature for Machine Translation System Combination. In *Workshop on Statistical Machine Translation*, pp. 467–476, Lisboa, Portugal, Sept. 2015.

[Gale 95] W.A. Gale: Good-Turing smoothing without tears. *Journal of Quantitative Linguistics*, Vol. 2, 1995.

[Galley & Manning 08] M. Galley, C.D. Manning: A Simple and Effective Hierarchical Phrase Reordering Model. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pp. 848–856, Honolulu, HI, USA, October 2008.

[Ganchev & Graça$^+$ 10] K. Ganchev, J.a. Graça, J. Gillenwater, B. Taskar: Posterior Regularization for Structured Latent Variable Models. *J. Mach. Learn. Res.*, Vol. 11, pp. 2001–2049, Aug. 2010.

[Garg & Peitz$^+$ 19] S. Garg, S. Peitz, U. Nallasamy, M. Paulik: Jointly Learning to Align and Translate with Transformer Models. In *Conference on Empirical Methods in Natural Language Processing*, Hong Kong, China, Nov. 2019.

[Gehring & Auli$^+$ 17a] J. Gehring, M. Auli, D. Grangier, Y. Dauphin: A Convolutional Encoder Model for Neural Machine Translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 123–135, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[Gehring & Auli$^+$ 17b] J. Gehring, M. Auli, D. Grangier, D. Yarats, Y.N. Dauphin: Convolutional Sequence to Sequence Learning, 2017.

[Gidon & Zolnik$^+$ 20] A. Gidon, T.A. Zolnik, P. Fidzinski, F. Bolduan, A. Papoutsi, P. Poirazi, M. Holtkamp, I. Vida, M.E. Larkum: Dendritic action potentials and computation in human layer 2/3 cortical neurons. *Science*, Vol. 367, No. 6473, pp. 83–87, 2020.

[Guta & Alkhouli$^+$ 15] A. Guta, T. Alkhouli, J.T. Peter, J. Wuebker, H. Ney: A Comparison between Count and Neural Network Models Based on Joint Translation and Reordering Sequences. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1401–1411, Lisbon, Portugal, Sept. 2015.

[Gutmann & Hyvärinen 10] M. Gutmann, A. Hyvärinen: Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Y.W. Teh, M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Vol. 9 of *Proceedings of Machine Learning Research*, pp. 297–304, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[Heafield 11] K. Heafield: KenLM: Faster and Smaller Language Model Queries. In *Conference on Empirical Methods in Natural Language Processing*, pp. 187–197, Edinburgh, Scotland, United Kingdom, July 2011.

[Heafield & Pouzyrevsky$^+$ 13] K. Heafield, I. Pouzyrevsky, J.H. Clark, P. Koehn: Scalable Modified Kneser-Ney Language Model Estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 690–696, Sofia, Bulgaria, Aug. 2013. Association for Computational Linguistics.

[Hinton & Osindero$^+$ 06] G.E. Hinton, S. Osindero, Y.W. Teh: A fast learning algorithm for deep belief nets. *Neural computation*, Vol. 18, No. 7, pp. 1527–1554, 2006.

[Hinton & Srivastava$^+$ 12] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. Salakhutdinov: Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, Vol. abs/1207.0580, 2012.

[Hochreiter & Bengio+ 01] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S.C. Kremer, J.F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*, pp. 237–244. IEEE Press, 2001.

[Hochreiter & Schmidhuber 97] S. Hochreiter, J. Schmidhuber: Long Short-Term Memory. *Neural Comput.*, Vol. 9, No. 8, pp. 1735–1780, Nov. 1997.

[Hu & Lan+ 15] X. Hu, X. Lan, H. Wu, H. Wang: Improved Beam Search with Constrained Softmax for NMT. 2015.

[Huck & Peter+ 12] M. Huck, J.T. Peter, M. Freitag, S. Peitz, H. Ney: Hierarchical Phrase-Based Translation with Jane 2. *The Prague Bulletin of Mathematical Linguistics*, Vol. 98, pp. 37–50, Oct. 2012.

[Hunt 90] M.J. Hunt: Figures of merit for assessing connected-word recognisers. *Speech Communication*, Vol. 9, No. 4, pp. 329–336, 1990.

[IBM 54] IBM: 701 Translator. Press release, Jan. 1954.

[International Telecommunication Union 19] International Telecommunication Union: Measuring digital development Facts and figures 2019. `https://www.itu.int/en/ITU-D/Statistics/Documents/facts/FactsFigures2019.pdf`, 2019. [Online; accessed 20-January-2020].

[Irie & Schlüter+ 15] K. Irie, R. Schlüter, H. Ney: Bag-of-Words Input for Long History Representation in Neural Network-based Language Models for Speech Recognition. In *Proceedings of the 16th Annual Conference of International Speech Communication Association*, pp. 2371–2375, Dresden, Germany, September 2015.

[Irie & Tüske+ 16] K. Irie, Z. Tüske, T. Alkhouli, R. Schlüter, H. Ney: LSTM, GRU, Highway and a Bit of Attention: An Empirical Overview for Language Modeling in Speech Recognition. In *Interspeech*, pp. 3519–3523, San Francisco, CA, USA, Sept. 2016.

[Jean & Cho+ 15] S. Jean, K. Cho, R. Memisevic, Y. Bengio: On Using Very Large Target Vocabulary for Neural Machine Translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1–10, Beijing, China, July 2015. Association for Computational Linguistics.

[Junczys-Dowmunt & Dwojak+ 16] M. Junczys-Dowmunt, T. Dwojak, R. Sennrich: The AMU-UEDIN Submission to the WMT16 News Translation Task: Attention-based NMT Models as Feature Functions in Phrase-based SMT, 2016.

[Kalchbrenner & Grefenstette+ 14] N. Kalchbrenner, E. Grefenstette, P. Blunsom: A Convolutional Neural Network for Modelling Sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pp. 655–665, Baltimore, MD, USA, June 2014.

[Katz 87] S.M. Katz: Estimation of probabilities from sparse data for the language model component of a speech recognizer. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, pp. 400–401, 1987.

[Khomenko & Shyshkov+ 17] V. Khomenko, O. Shyshkov, O. Radyvonenko, K. Bokhan: Accelerating recurrent neural network training using sequence bucketing and multi-GPU data parallelization. *CoRR*, Vol. abs/1708.05604, 2017.

[Kingma & Ba 14] D.P. Kingma, J. Ba: Adam: A Method for Stochastic Optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[Kneser & Ney 95] R. Kneser, H. Ney: Improved backing-off for M-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, Vol. 1, pp. 181 –184 vol.1, may 1995.

[Kocmi & Bojar 17] T. Kocmi, O. Bojar: Curriculum Learning and Minibatch Bucketing in Neural Machine Translation, 2017.

[Koehn 05] P. Koehn: Europarl: A parallel corpus for statistical machine translation. In *MT summit*, Vol. 5, pp. 79–86, 2005.

[Koehn & Hoang+ 07] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, E. Herbst: Moses: Open Source Toolkit for Statistical Machine Translation. In *Annual Meeting of the Association for Computational Linguistics*, pp. 177–180, Prague, Czech Republic, June 2007.

[Koehn & Och+ 03] P. Koehn, F.J. Och, D. Marcu: Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pp. 48–54, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[Krogh & Hertz 91] A. Krogh, J.A. Hertz: A Simple Weight Decay Can Improve Generalization. In *Proceedings of the 4th International Conference on Neural Information Processing Systems*, NIPS'91, pp. 950–957, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[Kudo & Richardson 18] T. Kudo, J. Richardson: SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71, Brussels, Belgium, Nov. 2018. Association for Computational Linguistics.

[Lehnen & Peter+ 13] P. Lehnen, J.T. Peter, J. Wuebker, S. Peitz, H. Ney: (Hidden) Conditional Random Fields Using Intermediate Classes for Statistical Machine Translation. In *Machine Translation Summit*, pp. 151–158, Nice, France, Sept. 2013.

[Leusch & Freitag+ 11] G. Leusch, M. Freitag, H. Ney: The RWTH System Combination System for WMT 2011. In *EMNLP 2011 Sixth Workshop on Statistical Machine Translation*, pp. 152–158, Edinburgh, UK, July 2011.

[Lewis & Simons+ 15] M.P. Lewis, G.F. Simons, C.D. Fennig: *Ethnologue: Languages of the World*, Vol. 18. SIL International, Dallas, TX, USA, 2015.

[Liew & Atkinson+ 16] C.S. Liew, M.P. Atkinson, M. Galea, T.F. Ang, P. Martin, J.I.V. Hemert: Scientific Workflows: Moving Across Paradigms. *ACM Comput. Surv.*, Vol. 49, No. 4, pp. 66:1–66:39, Dec. 2016.

[Lin 04] C.Y. Lin: Rouge: a package for automatic evaluation of summaries. pp. 25–26, 2004.

[Ludäscher & Altintas+ 06] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A. Lee, J. Tao, Y. Zhao: Scientific Workflow Management and the Kepler System: Research Articles. *Concurr. Comput. : Pract. Exper.*, Vol. 18, No. 10, pp. 1039–1065, Aug. 2006.

[Luong & Pham+ 15] M.T. Luong, H. Pham, C.D. Manning: Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[Luong & Sutskever+ 15] T. Luong, I. Sutskever, Q. Le, O. Vinyals, W. Zaremba: Addressing the Rare Word Problem in Neural Machine Translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 11–19, Beijing, China, July 2015. Association for Computational Linguistics.

[Manning & Schütze 99] C. Manning, H. Schütze: *Foundations of statistical natural language processing*. MIT Press, Cambridge, MA, 1999.

[McCulloch & Pitts 43] W. McCulloch, W. Pitts: A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Vol. 5, No. 4, pp. 115–133, December 1943.

[Meng & Lu+ 16] F. Meng, Z. Lu, H. Li, Q. Liu: Interactive Attention for Neural Machine Translation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pp. 2174–2185, Osaka, Japan, Dec. 2016. The COLING 2016 Organizing Committee.

[Mi & Sankaran+ 16] H. Mi, B. Sankaran, Z. Wang, A. Ittycheriah: A Coverage Embedding Model for Neural Machine Translation. *arXiv preprint arXiv:1605.03148*, Vol. , 2016.

[Mi & Wang+ 16a] H. Mi, Z. Wang, A. Ittycheriah: Supervised Attentions for Neural Machine Translation. *arXiv preprint arXiv:1608.00112*, Vol. , 2016.

[Mi & Wang+ 16b] H. Mi, Z. Wang, A. Ittycheriah: Vocabulary Manipulation for Neural Machine Translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 124–129, Berlin, Germany, Aug. 2016. Association for Computational Linguistics.

[Mikolov & Joulin+ 15] T. Mikolov, A. Joulin, S. Chopra, M. Mathieu, M. Ranzato: Learning Longer Memory in Recurrent Neural Networks. In *Proceedings of the 3rd International Conference on Learning Representations*, San Diego, CA, USA, May 2015.

[Mikolov & Karafiát+ 10] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, S. Khudanpur: Recurrent neural network based language model. In T. Kobayashi, K. Hirose, S. Nakamura, editors, *INTERSPEECH*, pp. 1045–1048. ISCA, 2010.

[Mikolov & Kombrink+ 11] T. Mikolov, S. Kombrink, A. Deoras, L. Burget, J.H. Cernocky: RNNLM - Recurrent Neural Network Language Modeling Toolkit. In *IEEE Automatic Speech Recognition and Understanding Workshop*, December 2011.

[Mnih & Teh 12] A. Mnih, Y.W. Teh: A Fast and Simple Algorithm for Training Neural Probabilistic Language Models. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pp. 419–426, USA, 2012. Omnipress.

[Moore & Lewis 10] R.C. Moore, W. Lewis: Intelligent Selection of Language Model Training Data. In *Annual Meeting of the Association for Computational Linguistics*, pp. 220–224, Uppsala, Sweden, July 2010.

[Morin & Bengio 05] F. Morin, Y. Bengio: Hierarchical probabilistic neural network language model. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*, pp. 246–252, Bridgetown, Barbados, January 2005.

[Nakamura & Shikano 89] M. Nakamura, M. Shikano: A study of English word category prediction based on neural networks. Vol. 2, pp. 731 – 734 vol.2, 06 1989.

[Nix 16] A. Nix: Alignment Methods for Attention-based Neural Machine Translation. Master's thesis, RWTH Aachen University, Department of Computer Science, Sept. 2016.

[Och 03] F.J. Och: Minimum Error Rate Training in Statistical Machine Translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, ACL '03, pp. 160–167, Sapporo, Japan, July 2003. Association for Computational Linguistics.

[Och & Ney 02] F.J. Och, H. Ney: Discriminative Training and Maximum Entropy Models for Statistical Machine Translation. In *Annual Meeting of the Association for Computational Linguistics*, pp. 295–302, Philadelphia, PA, USA, July 2002.

[Och & Ney 03] F.J. Och, H. Ney: A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, Vol. 29, pp. 19–51, March 2003.

[Och & Ney 04] F. Och, H. Ney: The Alignment Template Approach to Statistical Machine Translation. In *Proceedings of the Association for Computational Linguistics*, pp. 417–449, Barcelona, July 2004.

[Och & Tillmann$^+$ 99] F.J. Och, C. Tillmann, H. Ney: Improved Alignment models for Statistical Machine Translation. In *Conference on Empirical Methods in Natural Language Processing*, pp. 20–28, University of Maryland, College Park, MD, USA, June 1999.

[Papineni & Roukos$^+$ 02] K. Papineni, S. Roukos, T. Ward, W.J. Zhu: BLEU: A Method for Automatic Evaluation of Machine Translation. In *Annual Meeting of the Association for Computational Linguistics*, pp. 311–318, Philadelphia, PA, USA, July 2002. Association for Computational Linguistics.

[Pascanu & Mikolov$^+$ 13] R. Pascanu, T. Mikolov, Y. Bengio: On the Difficulty of Training Recurrent Neural Networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pp. III–1310–III–1318. JMLR.org, 2013.

[Peitz & Mansour$^+$ 13] S. Peitz, S. Mansour, J.T. Peter, C. Schmidt, J. Wuebker, M. Huck, M. Freitag, H. Ney: The RWTH Aachen Machine Translation System for WMT 2013. In *Workshop on Statistical Machine Translation*, pp. 193–199, Sofia, Bulgaria, Aug. 2013.

[Pérez & Granger 07] F. Pérez, B.E. Granger: IPython: a System for Interactive Scientific Computing. *Computing in Science and Engineering*, Vol. 9, No. 3, pp. 21–29, May 2007.

[Peter & Alkhouli$^+$ 16a] J.T. Peter, T. Alkhouli, A. Guta, N. Hermann: The RWTH Aachen University English-Romanian Machine Translation System for WMT 2016. In *ACL 2016 First Conference on Machine Translation*, 356–361, Berlin, Germany, Aug. 2016.

[Peter & Alkhouli$^+$ 16b] J.T. Peter, T. Alkhouli, H. Ney, M. Huck, F. Braune, A. Fraser, A. Tamchyna, O. Bojar, B. Haddow, R. Sennrich, F. Blain, L. Specia, J. Niehues, A. Waibel, A. Allauzen, L. Aufrant, F. Burlot, E. Knyazeva, T. Lavergne, F. Yvon, S. Frank, M. Pinnis: The QT21/HimL Combined Machine Translation System. In *ACL 2016 First Conference on Machine Translation*, 344–355, Berlin, Germany, Aug. 2016.

[Peter & Beck+ 18] J.T. Peter, E. Beck, H. Ney: Sisyphus, a Workflow Manager Designed for Machine Translation and Automatic Speech Recognition. In *Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, Nov. 2018.

[Peter & Guta+ 16] J.T. Peter, A. Guta, N. Rossenbach, M. Graça, H. Ney: The RWTH Aachen Machine Translation System for IWSLT 2016. In *International Workshop on Spoken Language Translation*, Seattle, USA, Dec. 2016.

[Peter & Guta+ 17] J.T. Peter, A. Guta, T. Alkhouli, P. Bahar, J. Rosendahl, N. Rossenbach, M. Graça, N. Hermann: The RWTH Aachen University English-German and German-English Machine Translation System for WMT 2017. In *EMNLP 2017 Second Conference on Machine Translation*, Copenhagen, Denmark, Sept. 2017.

[Peter & Ney+ 17] J.T. Peter, H. Ney, O. Bojar, N.Q. Pham, J. Niehues, A. Waibel, F. Burlot, F. Yvon, M. Pinnis, V. Šics, J. Bastings, M. Rios, W. Aziz, P. Williams, F. Blain, L. Specia: The QT21 Combined Machine Translation System for English to Latvian. In *EMNLP 2017 Second Conference on Machine Translation*, Copenhagen, Denmark, Sept. 2017.

[Peter & Nix+ 17] J.T. Peter, A.N. Nix, H. Ney: Generating Alignments Using Target Foresight in Attention-Based Neural Machine Translation. In *Conference of the European Association for Machine Translation*, pp. 27–36, Prague, Czech Republic, June 2017.

[Peter & Toutounchi+ 15a] J.T. Peter, F. Toutounchi, S. Peitz, P. Bahar, A. Guta, H. Ney: The RWTH Aachen German to English MT System for IWSLT 2015. In *International Workshop on Spoken Language Translation*, pp. 15–22, Da Nang, Vietnam, Dec. 2015.

[Peter & Toutounchi+ 15b] J.T. Peter, F. Toutounchi, J. Wuebker, H. Ney: The RWTH Aachen German-English Machine Translation System for WMT 2015. In *EMNLP 2015 Tenth Workshop on Statistical Machine Translation*, 158–163, Lisbon, Portugal, Sept. 2015.

[Peter & Wang+ 16] J.T. Peter, W. Wang, H. Ney: Exponentially Decaying Bag-of-Words Input Features for Feed-Forward Neural Network in Statistical Machine Translation. In *Annual Meeting of the Assoc. for Computational Linguistics*, pp. 293–298, Berlin, Germany, Aug. 2016.

[Popović & Ney 06] M. Popović, H. Ney: POS-based Word Reorderings for Statistical Machine Translation. In *International Conference on Language Resources and Evaluation*, pp. 1278–1283, Genoa, Italy, May 2006.

[Robinson & Fallside 87] A. Robinson, F. Fallside: *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering, 1987.

[Rochester & Holland+ 56] N. Rochester, J.H. Holland, L.H. Haibt, W.L. Duda: Tests on a cell assembly theory of the action of the brain, using a large digital computer. *IRE Trans. Information Theory*, Vol. 2, No. 3, pp. 80–93, 1956.

[Rosendahl 16] H. Rosendahl: Analysing Attention-based Neural Machine Translation. Master's thesis, RWTH Aachen University, Department of Computer Science, Sept. 2016.

[Rossenbach 16] N. Rossenbach: Training Neural Networks on Parallel Data without Alignment. Master's thesis, RWTH Aachen University, Department of Computer Science, March 2016.

[Rumelhart & Hinton+ 86] D.E. Rumelhart, G.E. Hinton, R.J. Williams: Learning representations by back-propagating errors. *Nature*, Vol. 323, pp. 533 EP –, 10 1986.

[Sankaran & Mi+ 16] B. Sankaran, H. Mi, Y. Al-Onaizan, A. Ittycheriah: Temporal Attention Model for Neural Machine Translation. *arXiv preprint arXiv:1608.02927*, Vol. , 2016.

[Schwenk 12] H. Schwenk: Continuous Space Translation Models for Phrase-Based Statistical Machine Translation. In *Proceedings of the 24th International Conference on Computational Linguistics*, pp. 1071–1080, Mumbai, India, December 2012.

[Schwenk & Déchelotte+ 06] H. Schwenk, D. Déchelotte, J.L. Gauvain: Continuous Space Language Models for Statistical Machine Translation. In *Proceedings of the 44th Annual Meeting of the International Committee on Computational Linguistics and the Association for Computational Linguistics*, pp. 723–730, Sydney, Australia, July 2006.

[Seide & Li+ 11] F. Seide, G. Li, D. Yu: Conversational Speech Transcription Using Context-Dependent Deep Neural Networks. In *Interspeech*, pp. 437–440, 2011.

[Sennrich & Haddow+ 15] R. Sennrich, B. Haddow, A. Birch: Neural Machine Translation of Rare Words with Subword Units. *CoRR*, Vol. abs/1508.07909, 2015.

[Sennrich & Haddow+ 16] R. Sennrich, B. Haddow, A. Birch: Improving Neural Machine Translation Models with Monolingual Data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 86–96, Berlin, Germany, Aug. 2016. Association for Computational Linguistics.

[Snover & Dorr+ 05] M. Snover, B. Dorr, R. Schwartz, J. Makhoul, L. Micciula, R. Weischedel: A Study of Translation Error Rate with Targeted Human Annotation. Technical Report LAMP-TR-126,CS-TR-4755,UMIACS-TR-2005-58, University of Maryland, College Park and BBN Technologies*, July 2005.

[Snover & Dorr+ 06] M. Snover, B. Dorr, R. Schwartz, L. Micciulla, J. Makhoul: A study of translation edit rate with targeted human annotation. In *In Proceedings of Association for Machine Translation in the Americas*, Vol. 200, pp. 223–231, Cambridge, MA, USA, August 2006.

[Son & Allauzen+ 12] L.H. Son, A. Allauzen, F. Yvon: Continuous Space Translation Models with Neural Networks. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL HLT '12, pp. 39–48, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[Srivastava & Greff+ 15] R.K. Srivastava, K. Greff, J. Schmidhuber: Training Very Deep Networks. In C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pp. 2377–2385. Curran Associates, Inc., 2015.

[Srivastava & Hinton+ 14] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov: Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, Vol. 15, No. 1, pp. 1929–1958, 2014.

[Stanojevic & Sima'an 14] M. Stanojevic, K. Sima'an: BEER: BEtter Evaluation as Ranking. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pp. 414–419. Association for Computational Linguistics, 2014.

[Stolcke 02] A. Stolcke: SRILM - An Extensible Language Modeling Toolkit. In *International Conference on Spoken Language Processing*, pp. 901–904, Denver, CO, USA, Sept. 2002.

[Sundermeyer & Alkhouli+ 14] M. Sundermeyer, T. Alkhouli, J. Wuebker, H. Ney: Translation Modeling with Bidirectional Recurrent Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 14–25, Doha, Qatar, October 2014.

[Sutskever & Vinyals+ 14] I. Sutskever, O. Vinyals, Q.V. Le: Sequence to Sequence Learning with Neural Networks. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pp. 3104–3112. Curran Associates, Inc., 2014.

[Tamura & Watanabe+ 14] A. Tamura, T. Watanabe, E. Sumita: Recurrent Neural Networks for Word Alignment Model. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1470–1480, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[Tu & Lu+ 16] Z. Tu, Z. Lu, Y. Liu, X. Liu, H. Li: Modeling Coverage for Neural Machine Translation. In *54th Annual Meeting of the Association for Computational Linguistics*, 2016.

[Van Merriënboer & Bahdanau+ 15] B. Van Merriënboer, D. Bahdanau, V. Dumoulin, D. Serdyuk, D. Warde-Farley, J. Chorowski, Y. Bengio: Blocks and fuel: Frameworks for deep learning. *arXiv preprint arXiv:1506.00619*, Vol. , 2015.

[Vaswani & Shazeer+ 17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L.u. Kaiser, I. Polosukhin: Attention is All you Need. In I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pp. 5998–6008. Curran Associates, Inc., 2017.

[Vaswani & Zhao+ 13] A. Vaswani, Y. Zhao, V. Fossum, D. Chiang: Decoding with Large-Scale Neural Language Models Improves Translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1387–1392, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[Vilar & Popovic+ 06] D. Vilar, M. Popovic, H. Ney: AER: do we need to "improve" our alignments? In *IWSLT*, pp. 205–212, 2006.

[Vilar & Stein+ 10] D. Vilar, D. Stein, M. Huck, H. Ney: Jane: Open Source Hierarchical Translation, Extended with Reordering and Lexicon Models. In *ACL 2010 Joint Fifth Workshop on Statistical Machine Translation and Metrics MATR*, pp. 262–270, Uppsala, Sweden, July 2010.

[Vogel & Ney+ 96] S. Vogel, H. Ney, C. Tillmann: HMM-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics - Volume 2*, COLING '96, pp. 836–841, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics.

[Wang 15] W. Wang: Investigation on Neural Network Inputs and Structures for Statistical Machine Translation. Master's thesis, RWTH Aachen University, Department of Computer Science, Nov. 2015.

[Wang & Lu+ 16] M. Wang, Z. Lu, H. Li, Q. Liu: Memory-enhanced Decoder for Neural Machine Translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 278–286, Austin, Texas, Nov. 2016. Association for Computational Linguistics.

[Wang & Peter+ 16] W. Wang, J.T. Peter, H. Rosendahl, H. Ney: CharacTER: Translation Edit Rate on Character Level. In *ACL 2016 First Conference on Machine Translation*, pp. 505–510, Berlin, Germany, Aug. 2016.

[Werbos 75] P. Werbos: *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University, 1975.

[Werbos 88] P.J. Werbos: Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, Vol. 1, No. 4, pp. 339 – 356, May 1988.

[Werbos 90] P.J. Werbos: Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, Vol. 78, No. 10, pp. 1550–1560, October 1990.

[Williams & Zipser 95] R.J. Williams, D. Zipser: Backpropagation. chapter Gradient-based Learning Algorithms for Recurrent Networks and Their Computational Complexity, pp. 433–486. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1995.

[Wilson & Martinez 03] D.R. Wilson, T.R. Martinez: The General Inefficiency of Batch Training for Gradient Descent Learning. *Neural Netw.*, Vol. 16, No. 10, pp. 1429–1451, Dec. 2003.

[Wolstencroft & Haines$^+$ 13] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M.P. Balcazar Vargas, S. Sufi, C. Goble: The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research*, Vol. 41, No. W1, pp. W557–W561, 2013.

[Wozniak & Armstrong$^+$ 13] J. Wozniak, T. Armstrong, M. Wilde, D.S. Katz, E. Lusk, I. Foster: Swift/T: Large-scale Application Composition via Distributed-memory Dataflow Processing. 05 2013.

[Wu & Schuster$^+$ 16] 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, Łukasz 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, J. Dean: Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016.

[Wuebker & Huck$^+$ 12] J. Wuebker, M. Huck, S. Peitz, M. Nuhn, M. Freitag, J.T. Peter, S. Mansour, H. Ney: Jane 2: Open Source Phrase-based and Hierarchical Statistical Machine Translation. In *International Conference on Computational Linguistics*, pp. 483–491, Mumbai, India, December 2012.

[Wuebker & Peitz$^+$ 13a] J. Wuebker, S. Peitz, T. Alkhouli, J.T. Peter, M. Feng, M. Freitag, H. Ney: The RWTH Aachen Machine Translation Systems for IWSLT 2013. In *International Workshop on Spoken Language Translation*, pp. 88–93, Heidelberg, Germany, Dec. 2013.

[Wuebker & Peitz$^+$ 13b] J. Wuebker, S. Peitz, F. Rietig, H. Ney: Improving Statistical Machine Translation with Word Class Models. In *Conference on Empirical Methods in Natural Language Processing*, pp. 1377–1381, Seattle, WA, USA, Oct. 2013.

[Yang & Hu$^+$ 17] Z. Yang, Z. Hu, Y. Deng, C. Dyer, A. Smola: Neural Machine Translation with Recurrent Attention Modeling. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 383–387, Valencia, Spain, April 2017. Association for Computational Linguistics.

[Zeiler 12] M.D. Zeiler: ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, Vol. , 2012.

[Zens & Och+ 02] R. Zens, F.J. Och, H. Ney, L.F.I. Vi: Phrase-Based Statistical Machine Translation. pp. 18–32. Springer Verlag, 2002.

[Zeyer & Beck+ 17] A. Zeyer, E. Beck, R. Schlüter, H. Ney: CTC in the Context of Generalized Full-Sum HMM Training. In *Interspeech*, pp. 944–948, Stockholm, Sweden, Aug. 2017.

[Zhang & Wang+ 17] J. Zhang, M. Wang, Q. Liu, J. Zhou: Incorporating Word Reordering Knowledge into Attention-based Neural Machine Translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1524–1534, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[Zhang & Xiong+ 16] B. Zhang, D. Xiong, J. Su: Recurrent Neural Machine Translation. *arXiv preprint arXiv:1607.08725*, Vol. , 2016.