Alignment-Based Neural Networks
for 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

M.Sc. RWTH
Tamer Ahmed Najeb Alkhouli
aus Jerusalem, Palästina

Berichter: Prof. Dr.-Ing. Hermann Ney
Prof. Dr. Khalil Sima’an

Tag der mündlichen Prüfung: 06. Juli 2020

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek online verfügbar.
“Real knowledge is to know the extent of one’s ignorance.”
- Confucius
ACKNOWLEDGMENTS

I am grateful to many people who helped me one way or another during my PhD journey. First and foremost, I would like to thank Prof. Dr-Ing. Hermann Ney, for allowing me the great opportunity of working at his chair, for teaching me the fundamentals of statistical pattern recognition, and for inspiring so many ideas through his vision.

I would also like to thank Prof. Dr. Khalil Sima’an for his interest in reviewing my dissertation, and accepting to be my second examiner.

I owe a lot of the experience I gained to my colleagues at the Human Language Technology and Pattern Recognition group. I had the excellent chance of working with Martin Sundermeyer and Joern Wuebker. I learned a great deal from their experimental design, critical thinking, and scientific writing. My thanks go to Saab Mansour, for guiding me through building competitive statistical machine translation systems. I would also like to extend my gratitude to Andreas Guta, for his insights on word alignment and phrase-based systems; Jan-Thorsten Peter, for the fruitful conversations about neural models; Malte Nuhn, for expressing genuine interest in my ideas; Parnia Bahar and Weiyue Wang, for the discussions on alignment-based neural machine translation. My thanks also extend to the rest of my machine translation colleagues: Stephan Peitz, Markus Freitag, Julian Schamper, Jan Rosendahl, Yunsu Kim, Yingbo Gao, and Christian Herold. Thank you for the stimulating discussions throughout my time at the chair.

I am lucky to have worked with Albert Zeyer and Kazuki Irie, who always managed to set the bar high, ensuring a competitive environment. I would also like to mention Priv.-Doz. Dr. rer. nat. Ralf Schlüter for inviting me to assist with the automatic speech recognition course.

I was fortunate to supervise talented students during their Bachelor and Master theses. Their work significantly influenced the direction of this dissertation. Thanks Felix Rietig, Gabriel Bretschner, and Mohammed Hethnawi. I would also like to thank Leonard Dahlmann and Munkhzul Erdenebash. It has been a pleasure working with all of you.

I would also like to acknowledge our system administrators Stefan, Kai, Jan R., JTP, Pavel, Weiyue, and Thomas, for making sure the infrastructure was up and running. Otherwise, this dissertation would not have seen the light. I would also like to thank our secretaries Andrea, Stephanie, Dhenya and Anna Eva, for their constant help and efficiency.

I would like to thank the following people who proofread my dissertation despite their tight schedules: Parnia Bahar, Weiyue Wang, Christian Herold, Yingbo Gao, Yunsu Kim, Jan Thorsten-Peter, Kazuki Irie, and Jan Rosendahl.

Thanks to Murat Akbacak for mentoring me during my internship at Apple. It was great working with you.

I would like to mention Patrick L., Patrick D., Minwei, Matthias, Jens, Simon, Pavel, Zoltán, Maldi, Amr, David, Christoph S., Mahaboob, Muhammad, Farzad, Michel, Markus K., Markus N., Harald, Oscar, Tobias, Christoph L., Volker, and Eugen. It was fun working with all of you.

I am forever indebted to my parents for my upbringing which set the path that culminated in this work. Finally, my deepest thanks goes to my wife Nur, together with whom I embarked on our research journey. Thanks for accommodating my deadlines and travels despite your packed schedule. I owe this work first and foremost to you.
Abstract

After more than a decade of phrase-based systems dominating the scene of machine translation, neural machine translation has emerged as the new machine translation paradigm. Not only does state-of-the-art neural machine translation demonstrate superior performance compared to conventional phrase-based systems, but it also presents an elegant end-to-end model that captures complex dependencies between source and target words. Neural machine translation offers a simpler modeling pipeline, making its adoption appealing both for practical and scientific reasons. Concepts like word alignment, which is a core component of phrase-based systems, are no longer required in neural machine translation. While this simplicity is viewed as an advantage, disregarding word alignment can come at the cost of having less controllable translation. Phrase-based systems generate translation composed of word sequences that also occur in the training data. On the other hand, neural machine translation is more flexible to generate translation without exact correspondence in the training data. This aspect enables such models to generate more fluent output, but it also makes translation free of pre-defined constraints. The lack of an explicit word alignment makes it potentially harder to relate generated target words to the source words. With the wider deployment of neural machine translation in commercial products, the demand is increasing for giving users more control over generated translation, such as enforcing or excluding translation of certain terms.

This dissertation aims to take a step towards addressing controllability in neural machine translation. We introduce alignment as a latent variable to neural network models, and describe an alignment-based framework for neural machine translation. The models are inspired by conventional IBM and hidden Markov models that are used to generate word alignment for phrase-based systems. However, our models derive from recent neural network architectures that are able to capture more complex dependencies. In this sense, this work can be viewed as an attempt to bridge the gap between conventional statistical machine translation and neural machine translation. We demonstrate that introducing alignment explicitly maintains neural machine translation performance, while making the models more explainable by improving the alignment quality. We show that such improved alignment can be beneficial for real tasks, where the user desires to influence the translation output.

We also introduce recurrent neural networks to phrase-based systems in two different ways. We propose a method to integrate complex recurrent models, which capture long-range context, into the phrase-based framework, which considers short context only. We also use neural networks to rescore phrase-based translation candidates, and evaluate that in comparison to the direct integration approach.
KURZFASSUNG

CONTENTS

1 Introduction ......................................................... 1
  1.1 Machine Translation ........................................... 1
  1.2 Statistical Machine Translation ............................... 1
  1.3 Neural Machine Translation .................................... 2
  1.4 About This Thesis .............................................. 2
  1.5 Publications ................................................... 3

2 Scientific Goals ................................................... 5

3 Preliminaries ...................................................... 7
  3.1 Terminology and Notation ..................................... 7
  3.2 Statistical Machine Translation ............................... 8
    3.2.1 Log-Linear Modeling .................................. 8
    3.2.2 Word Alignment ........................................ 9
  3.3 Evaluation Measures ......................................... 11
    3.3.1 BLEU .................................................. 11
    3.3.2 TER ................................................... 11

4 Neural Networks for Machine Translation ......................... 13
  4.1 Introduction .................................................. 13
  4.2 State of The Art .............................................. 14
  4.3 Terminology ................................................... 15
  4.4 Neural Network Layers ....................................... 16
    4.4.1 Input Layer ............................................. 16
    4.4.2 Embedding Layer ....................................... 16
    4.4.3 Feedforward Layer .................................... 16
    4.4.4 Recurrent Layer ....................................... 17
    4.4.5 Long Short-Term Memory Layer .......................... 18
    4.4.6 Softmax Layer ......................................... 19
    4.4.7 Output Layer ........................................... 19
  4.5 Neural Network Models For Machine Translation ............ 20
    4.5.1 Feedforward Lexicon Model ............................. 20
    4.5.2 Feedforward Alignment Model ........................... 21
    4.5.3 Recurrent Neural Network Models ....................... 22
    4.5.4 Recurrent Neural Network Language Models ............ 23
    4.5.5 Recurrent Neural Network Lexical Model ............... 24
    4.5.6 Bidirectional Recurrent Neural Network Lexical Model 26
    4.5.7 Bidirectional Recurrent Neural Network Alignment Model 30
5 Neural Machine Translation

5.1 Introduction .............................................. 47
5.2 State of The Art ......................................... 48
5.3 Model .................................................. 49
5.4 Subword Units ........................................... 50
5.5 Search .................................................. 50
5.6 Alignment-Based Neural Machine Translation ......................... 51
  5.6.1 Models .............................................. 53
  5.6.2 Training ............................................ 53
  5.6.3 Search .............................................. 59
  5.6.4 Pruning ............................................. 61
  5.6.5 Alignment Extraction ................................. 63
5.7 Experimental Evaluation ..................................... 64
  5.7.1 Setup ............................................... 64
  5.7.2 Alignment vs. Attention Systems ....................... 66
  5.7.3 Word vs. Byte-Pair-Encoded Subword Vocabulary ............. 69
  5.7.4 Feedforward vs. Recurrent Alignment Systems ............... 69
  5.7.5 Bidirectional Recurrent Lexical Model Variants ............... 72
  5.7.6 Alignment-Biased Attention ................................ 72
  5.7.7 Class-Factored Output Layer .......................... 73
  5.7.8 Model Weights ...................................... 75
  5.7.9 Alignment Variants .................................. 75
  5.7.10 Alignment Pruning ................................... 75
  5.7.11 Alignment Quality .................................. 77
  5.7.12 Dictionary Suggestions ................................ 78
  5.7.13 Forced Alignment Training ............................ 81
  5.7.14 Qualitative Analysis .................................. 84
5.8 Contributions ........................................... 85

6 Phrase-Based Machine Translation........................................ 89

6.1 Introduction ............................................. 89
6.2 State of The Art ......................................... 90
6.3 Model Definition ......................................... 91
  6.3.1 Phrase Translation Models ............................. 93
  6.3.2 Word Lexicon Models ................................ 93
  6.3.3 Phrase-Count Indicator Models ......................... 94
  6.3.4 Enhanced Low-Frequency Model ......................... 94
Translation refers to communicating the meaning of text written in a source natural language through text written in a target natural language. In a globalized world, translation is much needed to break barriers between people. The need for translation arises in official settings, such as that of the European Commission, which has a Directorate-General for Translation that is responsible for the largest translation service in the world, translating text from and out of the 24 official languages of the European Union. Thanks to the numerous social media platforms, the user is likely to encounter content written in a foreign language produced by other users, and the need for instant translation is nowadays more likely to arise. Since human translation is laborious and expensive, automatic translation, referred to as machine translation, plays a key role in unlocking foreign-language content to millions of users. Machine translation is not only applied to generate standalone translation, but also used to assist human translators in creating faster translations, which is referred to as computer-aided translation.

1.1 Machine Translation

Machine translation is the task of automatic conversion of text written in one natural language, called the source language, to text written in another natural language, the target language. We can describe three conceptually different approaches to machine translation [Vauquois 68]. First, direct translation from the source to the target language, where the problem is handled as strict low-level text-to-text conversion disregarding syntax and semantics. The second approach is the ‘transfer approach’, where conversion is performed via a transfer stage between abstract representations of the source and target text. This abstract representation is obtained by analyzing the source text. The source representation goes via a transfer stage to generate an abstract target representation, which is used to generate the final target text in a generation stage. This means that translating the text through this approach requires an analysis stage of the source text, a transfer stage, and a generation stage to create the target text. The third approach to machine translation aims at converting the source text to an interlingual representation that is language-independent, the target text is generated from this universal representation.

It can also be distinguished between rule-based and data-driven machine translation. Rule-based approaches focus on manually-created translation rules for a given language pair. This requires human knowledge and is often expensive to obtain. On the other hand, data-driven approaches like statistical machine translation do not require such human knowledge, but rely on data examples to model translation.

1.2 Statistical Machine Translation

Statistical machine translation is a data-driven approach that emerged in the late 1980s. The idea behind it is to have translation models that can be trained using source and target corpora.
The statistical models are used to translate the source text to target text without the need for manually created translation rules. In the literature, statistical machine translation often refers to word- and phrase-based models trained using the maximum-likelihood criterion. Early statistical machine translation systems were word-based, where each translation step consists of generating one word [Brown & Della Pietra+ 93, Vogel & Ney+ 96]. In the early 2000s, phrase-based systems were proposed [Zens & Och+ 02, Koehn & Och+ 03]. These systems became widely adopted as the state-of-the-art machine translation systems for over a decade. Afterwards, neural machine translation was introduced and became the dominant machine translation paradigm.

1.3 Neural Machine Translation

Neural machine translation is another data-driven approach. It refers to machine translation systems that use neural network models to generate translation, where the model receives the source sentence as input and outputs the target sentence [Kalchbrenner & Blunsom 13, Bahdanau & Cho+ 14, Sutskever & Vinyals+ 14, Cho & van Merrienboer+ 14a]. Early neural machine translation attempts started in 2013. By 2015, the research community had embraced neural machine translation as the new paradigm. In comparison to the phrase-based framework, neural machine translation offers a model that is trained end-to-end without requiring extra intermediate steps, such as word alignment. It also has more consistency between training and evaluation. Moreover, neural machine translation often demonstrates superior performance by a large margin compared to phrase-based systems, especially in cases where there is enough parallel training data. When human evaluators compare the output of the two systems, they often find neural translation more fluent than phrase-based translation.

Although neural models are statistical models, neural machine translation is often contrasted to statistical machine translation.

1.4 About This Thesis

This thesis is centered around a specific type of neural network models, namely alignment-based neural networks. Chapter 2 explains the scientific goals of this dissertation. Chapter 3 introduces the terminology, notation, and basic concepts of statistical machine translation. In Chapter 4, we give an overview of the basics of neural networks, and delve into specific neural models that are dependent on alignment. We explain the architecture of the different models used in the rest of this dissertation. The models range from simple feedforward networks to recurrent neural networks (RNN) and transformer networks that use multiple attention heads to implicitly model alignment. These models are applied in neural machine translation, which is described in Chapter 5. This chapter explains how to use the proposed models to build an alignment-based neural machine translation system. We also explain how to modify the search procedure to include these models, and introduce tricks to speed up translation. Chapter 6 explains how to use alignment-based neural networks within the phrase-based framework. We discuss first-pass integration where the models are included directly in search, and second-pass integration where the models are used to re-rank already generated translation candidates. Chapter 7 revisits the scientific achievements in light of the scientific goals, and Chapter 8 highlights the author’s individual contributions in contrast to team work.
1.5 Publications

The following scientific publications were published by the author at peer-reviewed conferences during the course of this dissertation:

- **Phrase-based machine translation**
  - [Sundermeyer & Alkhouli+ 14] Translation Modeling with Bidirectional Recurrent Neural Networks (EMNLP).
    The author designed and implemented the bidirectional lexicon model, and implemented N-best rescoring into the *ruthlin* neural toolkit.
    The author proposed a variant of the bidirectional lexicon model, and applied caching to integrate recurrent language and lexicon models into phrase-based search.

- **Neural machine translation**
    The author proposed the alignment-based neural machine translation framework using feedforward models.
  - [Alkhouli & Ney 17] Biasing Attention-Based Recurrent Neural Networks Using External Alignment Information (WMT).
    The author proposed biasing attention energies with external alignment information, and proposed a recurrent alignment model.
    The author helped with deriving the auxiliary function for the expectation-maximization training of feedforward models.
  - [Alkhouli & Bretschner+ 18] On The Alignment Problem In Multi-Head Attention-Based Neural Machine Translation (WMT).
    The author proposed adding an alignment head to the multi-head attention, and proposed a self-attentive alignment model. The author designed experiments to demonstrate the effectiveness of the alignment-based approach for dictionary integration.

This is a list of system papers co-authored by the author of this dissertation. These publications include models proposed and discussed in this work:

- **International evaluation campaigns**
    The author applied recurrent language and bidirectional lexicon models in N-best rescoring of phrase-based output. The models were part of a system combination for the English→Romanian task. The system combined all 11 participating systems and ranked 1st among them according to the BLEU score [Bojar & Chatterjee+ 16].
    The author applied recurrent language and bidirectional lexicon models in N-best rescoring of phrase-based output. The models were part of the RWTH system combination for the English→Romanian task. The system ranked 3rd among 12 systems according to the BLEU score [Bojar & Chatterjee+ 16].
1 Introduction

- [Peter & Guta+ 17] The RWTH Aachen University English-German and German-English Machine Translation System for WMT 2017 (WMT). The author applied bidirectional recurrent lexicon and alignment models in N-best rescoring of phrase-based output. The models were part of the RWTH system combination for the German→English task. The system ranked in the first cluster tied with 5 other systems according to the human evaluation [Bojar & Chatterjee+ 17]. There were 11 systems in total.

In addition, the author published the following papers which are not directly covered in this dissertation:

- **Phrase-based machine translation**
  - [Alkhouli & Guta+ 14] Vector Space Models for Phrase-Based Machine Translation (SSST).
  - [Guta & Alkhouli+ 15] A Comparison between Count and Neural Network Models Based on Joint Translation and Reordering Sequences (EMNLP).

- **Neural machine translation**

- **International evaluation campaigns**

- **Language modeling**

- **Toolkits**
  - [Zeyer & Alkhouli+ 18] RETURNN as A Generic Flexible Neural Toolkit with Application to Translation and Speech Recognition (ACL).

- **Error bounds**
2. Scientific Goals

In this thesis, we aim to pursue the following scientific goals:

- The move from phrase-based to neural machine translation was not smooth, but rather a radical jump highlighting a paradigm shift that got rid of most components of phrase-based systems. Neural machine translation has many appealing aspects. Unlike phrase-based systems, training neural machine translation models is usually done end-to-end without the need for cascaded ad-hoc training steps. On the other hand, neural machine translation has an explainability problem—it is not trivial, if at all possible, to determine why certain words are being generated. In phrase-based systems, the generation of phrases is limited to a large table of phrases, which makes it easier to determine the source phrase and override its translation if needed. In this thesis, we introduce alignment-based neural machine translation in Section 5.6, an alternative approach to standard neural machine translation. While we stick to neural models for their superior performance, we explicitly model word alignment. By introducing the alignment concept to neural models, we expect to better map target words to the source words used to generate them, hence making neural machine translation more explainable. This can be viewed as a gap-bridging approach positioned between the phrase-based paradigm that essentially relies on word alignment, and the neural machine translation paradigm. To this end, we design alignment-based neural network translation models (cf. Sections 4.5.1, 4.5.6, 4.7.1, and 4.7.2) and alignment models (cf. Sections 4.5.2, 4.5.7, and 4.7.3), and devise a search algorithm based on word alignment (cf. Section 5.6.3). Such explicit alignment modeling is lacking in state-of-the-art neural machine translation systems, which replace it with one or more attention components that are computed softly as part of the model [Bahdanau & Cho+ 14, Vaswani & Shazeer+ 17]. We aim to determine how alignment-based models compare to the corresponding neural machine translation systems in terms of translation quality (cf. Section 5.7.2).

- In addition to studying the performance of alignment-based neural machine translation in standard translation settings, we aim to explore the performance of the proposed systems for tasks that can benefit from the alignment information. Such tasks arise in commercial products where customers want to dictate their own translation overrides. We aim to investigate how alignment-based systems compare to standard systems for such tasks (cf. Sections 5.7.11-5.7.12).

- One appealing aspect of neural machine translation is that it is an end-to-end approach. We aim to study whether alignment-based systems can be trained using self-generated alignment, which lays the ground for end-to-end training of alignment-based systems, and achieves more consistency between training and decoding. We plan to study the prospects of forced-alignment training (Section 5.6.2) and its impact on the eventual performance of alignment-based systems (Section 5.7.13). To this end, we will introduce a forced-alignment training algorithm that alternates between computing alignment and neural training.
• We will investigate the use of neural networks in phrase-based systems (Chapter 6). We seek to design translation neural networks and integrate them into phrase-based systems. We will compare the use of neural networks using two main approaches: \( N \)-best rescoring, and direct decoder integration. \( N \)-best rescoring consists of generating translation candidates independent of the neural models first, then applying neural models to re-rank the generated candidates. On the other hand, direct decoder integration evaluates neural networks directly during phrase-based search, such that their scores are directly used to generate translation. We will introduce search algorithms that allow for direct integration of recurrent neural models in phrase-based search. Such direct decoder integration uses all available knowledge sources, including the neural models, to generate translation. \( N \)-best rescoring, on the other hand, limits the application of neural models to a limited selection of translation candidates, and is done as a second stage. We aim to compare the two approaches in terms of translation quality, and to provide error analysis of our direct integration approach.

• We seek to evaluate all our proposed methods on publicly available large-scale tasks, whether they are part of research projects or public evaluation campaigns. This allows for comparability with results from other groups.
3. PRELIMINARIES

In this chapter, we will introduce the main terminology and concepts used throughout this work.

3.1 Terminology and Notation

In this work, we discuss two different approaches to machine translation: phrase-based machine translation (PBT), and neural machine translation (NMT). We use the term decoder to refer to the algorithm that searches for the best translation. We use decoding and search interchangeably to denote the execution of the decoder algorithm. The input language to the machine translation engine is called the source language, and the output language is the target language. The ground-truth target translation is referred to as the reference. This can be created by human translators or automatically. Given a source sentence, the machine translation engine generates a hypothesis translation as a result of decoding. Statistical machine translation systems undergo training which refers to the parameter estimation part of building the system. This work makes use of supervised training which requires a parallel corpus, consisting of sentence-aligned source and target sentences. This is also referred to as a bilingual corpus. The target side of this corpus contains reference translations of the source sentences.

We distinguish between the training dataset (train), which is a large parallel corpus used to train the system, and the development dataset (dev), which is a small parallel corpus used for hyper-parameter tuning, or for learning rate scheduling while training neural networks. We use one or more test datasets (test) to evaluate the output of the translation system after training is complete. The test datasets are not exposed to the system, neither during training nor tuning. Typically, train is crawled from the web or multilingual government sources (e.g. European Parliament speeches) and then automatically aligned on the sentence level. dev and test are usually of high-quality human translation.

We will denote a source sequence of length \( J \) (i.e. containing \( J \) words) as \( F = f_1^J = f_1 f_2 \ldots f_J \). \( E = e_1^I = e_1 e_2 \ldots e_I \) is the corresponding target sequence of length \( I \). \( e_0 = \langle s \rangle \) is always fixed as the sentence begin symbol prepended to the target sentence. Historically, \( f \) stands for French or foreign and \( e \) for English. We use \( b_i^I = b_1 b_2 \ldots b_I \) to denote the word alignment path mapping target positions to source positions. \( b_i \) denotes the source position aligned to the target position \( i \). Similarly, \( a_j^I = a_1 a_2 \ldots a_J \), denotes the word alignment path mapping source positions to target positions. \( a_j \) aligns the source position \( j \) to the target position \( a_j \). This work is rooted in probability theory. We will distinguish between the true unknown probability distribution \( Pr(\cdot) \) and the model distribution \( p(\cdot) \).
3 Preliminaries

3.2 Statistical Machine Translation

Statistical machine translation (SMT) is a natural language processing (NLP) task that uses statistics and probability theory to solve translation. An SMT system is composed of one or several models. The model parameters are estimated during training using translation samples, which are sequences of source and target words or tokens. Training seeks to estimate a model that approximates the true unknown probability distribution. In other words, the model should approximate the true distribution that underlies the translation samples used to train it.

Formally, we use Bayes’ decision rule to obtain translation. Given a source sequence \( f_1^J = f_1 f_2 \ldots f_J \) to be translated, we look for the translation \( \hat{e}_1^I \) that has the maximum probability according to the true posterior distribution \( Pr(e_1^I | f_1^J) \)

\[
\begin{align*}
  f_1^J \rightarrow \hat{e}_1^I(f_1^J) &= \arg\max_{I, e_1^I} \{ Pr(e_1^I | f_1^J) \}.
\end{align*}
\]

Note that this maximization is also carried over the length of the translation \( I \), since the target length is unknown. Based on this, we can identify three main categorical problems of SMT, which cover all the topics that will be discussed in this dissertation:

- the **modeling problem** refers to designing models that capture structural dependencies in the training data in order to approximate the true posterior distribution;
- the **training problem** refers to estimating the model parameters using the training data;
- the **search problem** refers to carrying out the maximization in Equation 3.1.

3.2.1 Log-Linear Modeling

The posterior distribution can be decomposed using Bayes’ theorem into a language model \( Pr(e_1^I) \) and an inverse translation model \( Pr(f_1^J | e_1^I) \) as follows [Brown & Cocke + 90]:

\[
\begin{align*}
  f_1^J \rightarrow \hat{e}_1^I(f_1^J) &= \arg\max_{I, e_1^I} \{ Pr(e_1^I | f_1^J) \} = \arg\max_{I, e_1^I} \left\{ \frac{Pr(e_1^I) \cdot Pr(f_1^J | e_1^I)}{Pr(f_1^J)} \right\}
  = \arg\max_{I, e_1^I} \left\{ Pr(e_1^I) \cdot Pr(f_1^J | e_1^I) \right\}.
\end{align*}
\]

where we drop the denominator because the maximizing arguments are independent of it. This can be generalized to include more models using a log-linear model for the posterior distribution [Papineni & Roukos + 98, Och & Ney 02]

\[
Pr(e_1^I | f_1^J) = \frac{\exp \left( \sum_{m=1}^{M} \lambda_m h_m(e_1^I, f_1^J) \right)}{\sum_{I', e_1'^I} \exp \left( \sum_{m=1}^{M} \lambda_m h_m(e_1'^I, f_1^J) \right)},
\]

where \( M \) is the number of models, \( h_m(\cdot, \cdot) \) is the \( m \)-th model or feature for \( m = 1 \ldots M \), and \( \lambda_m \) is its corresponding scaling factor. The scaling factors in phrase-based systems are usually tuned on dev using e.g. minimum error rate training (MERT) [Och 03]. Substituting the log-linear model of Equation 3.4 in Equation 3.1 results in the following decision rule
Figure 3.1: Illustration of word alignment, where alignment is indicated by the shaded areas. A word can be aligned to a single or multiple words on the opposite side. Words can also be unaligned, such as the punctuation mark ‘.’ at the end of the target sentence.

\[
f_1^t \rightarrow \hat{e}_t^1(f_1^t) = \arg\max_{I,e_t^1} \left\{ \sum_{m=1}^{M} \lambda_m h_m(e_t^1,f_1^t) \right\}, \tag{3.5}
\]

where we again drop the denominator because it is constant with respect to the maximizing arguments. The exponential function is dropped since it is strictly monotonous and does not affect the maximizing argument.

Phrase-based machine translation uses the log-linear framework to include several phrase- and word-level models. Many of the models rely on the Viterbi word alignment, which will be discussed next.

### 3.2.2 Word Alignment

Word alignment refers to the word-level correspondence between words in the source and target sequences. Usually, parallel corpora are not annotated with word-level alignment; therefore, word alignment is computed automatically. Formally, the word alignment \( A \subseteq \{1, 2, \ldots, I\} \times \{1, 2, \ldots, J\} \) is defined as a relation over source indices \( j \in \{1, 2, \ldots, J\} \) and target indices \( i \in \{1, 2, \ldots, I\} \). The \( i \)-th target word \( e_i \) is aligned to the \( j \)-th source word \( f_j \) iff \((i, j) \in A\). An example word alignment is given in Figure 3.1.

Word alignment can be introduced as a latent or hidden variable sequence. In the following, we will assume that each source word \( f_j \) is aligned exactly to one target word \( e_i \). Let \( a_1^j = a_1a_2 \ldots a_J \) be the source-to-target word alignment sequence, where \( a_j = i \) indicates that the \( j \)-th source word is aligned to the \( i \)-th target word. We can introduce the alignment hidden sequence as follows.
where the inverse translation probability is decomposed into a length probability, an alignment probability and a lexical probability.

Word alignment is usually computed using multiple stages of IBM 1-5 [Brown & Della Pietra 93] and hidden Markov model (HMM) training [Vogel & Ney 96]. The models differ in the way they structure the dependencies.

- **IBM 1** has the weakest dependencies. It assumes a uniform alignment distribution and a zero-order lexical distribution

  \[
  Pr(a_j|a_{j-1}, f_{j-1}, e_i, J) = \frac{1}{i + 1} \\
  Pr(f_j|f_{j-1}, a_j, e_i, J) = p(f_j|a_j).
  \]

- **The HMM model** uses the same lexical model as IBM1, but it also makes a first-order hidden Markov assumption for the alignment probability

  \[
  Pr(a_j|a_{j-1}, f_{j-1}, e_i, J) = p(a_j|a_{j-1}, J, I).
  \]

- **IBM 4** has more complex dependencies. It makes use of the concept of fertility \( \varphi(e_i) \), which is the number of source words aligned to \( e_i \), where the source length \( J = \sum \varphi(e_i) \). It uses a more complex alignment model that is conditioned on source and target word classes. The alignment model has a first-order dependence on the \( j \)-axis, as opposed to the HMM model which has the first order dependence over the \( i \)-axis.

The models are trained with the EM algorithm [Dempster & Laird 77]. The Viterbi alignment \( \hat{a}_J^{e_1} \) of a sentence pair \((e_i, f_{i_1})\) is the alignment sequence that maximizes the alignment posterior distribution

\[
(e_i, f_{i_1}) \rightarrow \hat{a}_J^{e_1}(e_i, f_{i_1}) = \arg\max_{a_{i_1}^j} \{ Pr(a_{i_1}^j|e_{i_1}, f_{i_1}) \} \tag{3.8}
\]

\[
= \arg\max_{a_{i_1}^j} \{ Pr(f_{i_1}|a_{i_1}^j, e_{i_1}) \}. \tag{3.9}
\]

A common training setting is to start with IBM 1 training, since it is a convex optimization problem. This is followed by HMM training followed by IBM 4 training, where each stage is used to initialize the training of the next stage. In this dissertation, we use GIZA++ [Och & Ney 03] to compute the word alignment using the IBM 1/HMM/IBM 4 training scheme, where we perform 5 EM iterations in each stage. We compute source-to-target and target-to-source word alignment and combine them using the grow-diagonal-final-and heuristic [Koehn & Och 03].
3.3 Evaluation Measures

It is essential for feasible development of machine translation systems to have an automatic way of evaluating the output translation. To this day, evaluation measures for machine translation remain an open research question. This is because what constitutes correct translation is inherently difficult to answer. Humans themselves can disagree on what is a better translation when proposed with different translation candidates. Moreover, one source sentence can have multiple correct translations due to different word ordering and the use of synonyms. Nevertheless, we follow the research community and evaluate our systems in this dissertation using BLEU and TER, the two most common automatic evaluation metrics, where BLEU is a precision measure and TER is an error measure.

3.3.1 BLEU

The Bilingual Evaluation Understudy (BLEU) [Papineni & Roukos+ 02] is a precision measure. It is based on counting how often short word sequences in a given sentence occur in the corresponding reference sentence. BLEU uses the precision $\text{Prec}_n(e^I_1, \hat{e}^\hat{I}_1)$ between the output translation $e^I_1$, and the reference translation $\hat{e}^\hat{I}_1$, which is defined as follows:

$$\text{Prec}_n(e^I_1, \hat{e}^\hat{I}_1) := \frac{\sum w^n_1 \min\{C(w^n_1|e^I_1), C(w^n_1|\hat{e}^\hat{I}_1)\}}{\sum_{w^n_1 = I-n+1} C(w^n_1|e^I_1)}, \quad (3.10)$$

where $w^n_1 = w_1 w_2 ... w_n$ is the $n$-gram of $n$ consequent words, $C(w^n_1|e^I_1)$ is the number of occurrences of $w^n_1$ in the sentence $e^I_1$. The denominator is the total number of $n$-grams in the sentence $e^I_1$. BLEU also uses a brevity penalty $BP(I, \hat{I})$ that penalizes short hypotheses

$$BP(I, \hat{I}) := \begin{cases} 1 & \text{if } I \geq \hat{I}, \\ e^{(I-\hat{I})} & \text{if } I < \hat{I}. \end{cases} \quad (3.11)$$

BLEU is computed as a geometric mean of $n$-gram precisions and scaled by the brevity penalty as follows:

$$\text{BLEU}(e^I_1, \hat{e}^\hat{I}_1) := BP(I, \hat{I}) \cdot \prod_{n=1}^{4} \sqrt[n]{\text{Prec}_n(e^I_1, \hat{e}^\hat{I}_1)}. \quad (3.12)$$

We use document-level BLEU, where the $n$-gram counts are computed using the full dataset rather than a single sentence. Since it is a precision measure, higher BLEU values are better.

3.3.2 TER

The Translation Edit Rate (TER) [Snover & Dorr+ 06] is an error metric based on the Levenshtein distance [Levenshtein 66]. It counts the minimum number of edit operations required to change a hypothesis into the reference translation. The possible edit operations are substitution, deletion, insertion, and shift. Shifts involve shifting contiguous word sequences within the hypothesis. All types of edits are assigned equal cost. The number of edit operations is divided by the number of reference words

$$\text{TER}(e^I_1, \hat{e}^\hat{I}_1) = \frac{\min \# \text{ of edits to convert } e^I_1 \text{ to } \hat{e}^\hat{I}_1.}{\hat{I}}. \quad (3.13)$$
3 Preliminaries

We report document-level TER, where the minimum number of edit operations per-sentence is accumulated over the whole dataset, and normalized by the number of words in the reference dataset. Since TER is an error measure, lower TER values are better.
4. Neural Networks for Machine Translation

Neural network models gained increasing attention throughout the period of this dissertation. Having demonstrated strong improvements in other tasks including speech recognition and computer vision [Krizhevsky & Sutskever+ 12, Karpathy & Fei-Fei 15, Hinton & Deng+ 12, Mnih & Kavukcuoglu+ 15, Le 13, Karpathy & Toderici+ 14], the machine translation community started experimenting with neural models. It should be noted, however, that early ideas of using neural networks for machine translation can be traced back to many papers including [Allen 87, Castano & Casacuberta 97, Forcada & Ñeco 97].

Initially, neural models were introduced as complimentary models in phrase-based systems. Soon after, neural models become the core machine translation engines, the approach was named neural machine translation [Sutskever & Vinyals+ 14, Bahdanau & Cho+ 14, Cho & van Merrienboer+ 14a]. The approach appealed to the community not only because it simplifies the machine translation system by using a single model, but also because the model is trained end-to-end, as no extra intermediate steps are introduced. In comparison to phrase-based systems that necessarily use multiple models, and that have a relatively complex multi-stage training scheme, neural machine translation is simpler, cleaner, and even outperforms phrase-based systems. As of writing this dissertation, neural machine translation remains the dominant approach in machine translation.

This chapter explains the basics of neural networks that are relevant to the rest of the dissertation. We give a brief introduction and an overview of the state of the art in Sections 4.1 and 4.2, respectively. Section 4.3 introduces the terminology, and Section 4.4 explains different neural network layers. These layers are used as building blocks in the remaining sections. We will discuss different feedforward and recurrent neural network architectures for machine translation in Section 4.5. These models are relevant for neural machine translation (Chapter 5) and phrase-based systems (Chapter 6). We discuss recurrent attention-based and so-called transformer models in Section 4.6. Section 4.7 presents the combination of external alignment information with both types of models. These Sections are relevant to Chapters 5 and 6. Section 4.8 discusses the training procedure. Finally, we highlight our specific contributions in Section 4.9.

4.1 Introduction

Neural networks have become increasingly popular in natural language processing tasks [Mikolov & Karafiát+ 10, Sundermeyer & Schüting+ 12, Mikolov & Sutskever+ 13, Akbik & Blythe+ 18, Devlin & Chang+ 19]. They can be contrasted to count-based models that treat words are discrete events. Neural networks map discrete words into a continuous space, where each word is represented as a real-valued vector. These continuous representations are learned automatically as part of the training procedure of the neural network as a whole. This continuous
representation is specially appealing for modeling words, because it allows computing distances between them. The notion of distance can be used to determine whether words are semantically close to each other. In comparison, models based on discrete words cannot capture such semantic relationship between words. It is worth noting that there are approaches other than neural networks used to create continuous-space word representations, such as using co-occurrence counts [Lund & Burgess 96, Landauer & Dumais 97]. Such representations are sparse and high-dimensional, which can require an additional dimensionality reduction step, such as singular value decomposition (SVD). In this work, however, we only focus on neural network modeling, since this has been the most successful and widely-used approach in the natural language processing research community.

When moving beyond words to sequences of words, neural networks exhibit a generalization capability not available to count-based models. E.g., count-based language models apply sophisticated smoothing techniques such as backing off and discounting [Kneser & Ney 95, Ney & Essen 94], which is applied to assign non-zero probability values to sequences of words that were not part of the training data. Zero probability values are problematic in statistical data-driven methods that include multiple knowledge sources, or when applying the chain rule of probability, as multiplying by a zero probability leads to an overall zero score. Neural networks, on the other hand, are able to compute non-zero probabilities of word sequences not observed during training. This eliminates the need for explicit smoothing.

A neural network model is a function approximator that maps input values to output values. In this dissertation, the input to the neural network is one or multiple words, and the output is a probability distribution over discrete events. We use two output types in this work: words and alignment jumps. In both cases, the output is a normalized probability distribution. The rest of this chapter discusses different neural network architectures and how to train them. We focus on language and translation neural network models.

### 4.2 State of The Art

[Devlin & Zbib++ 14] propose a feedforward lexicon model that utilizes word alignment to select source words as input to the model. It uses a self-normalized output layer, where the training objective is augmented to produce approximately normalized scores at evaluation time, without the need to explicitly compute the normalization factor. In Section 4.5.1, we describe a model similar in structure, which is introduced in [Alkhouli & Bretschner++ 16]. Instead of the self-normalized output layer, the model has a class-factored output layer to reduce the cost of the softmax normalization factor. [Sundermeyer & Alkhouli++ 14] and [Alkhouli & Rietig++ 15] propose replacing the feedforward layers with recurrent layers to capture the complete target context. They also propose using bidirectional recurrent layers to compute source representations depending on the full source context. The models are described in Sections 4.5.5 and 4.5.6. [Yang & Liu++ 13] use neural network lexical and alignment models, but they give up the probabilistic interpretation and produce unnormalized scores instead. Furthermore, they model alignments using a simple distortion model that has no dependence on the lexical context. The models are used to produce new alignments which are used to train phrase-based systems. [Tamura & Watanabe++ 14] propose a lexicalized RNN alignment model. The model also produces non-probabilistic scores, and is used to generate word alignments used to train phrase-based systems. [Alkhouli & Bretschner++ 16] propose a feedforward alignment model to score relative jumps over the source positions. The model is described in Section 4.5.2. A recurrent version of the alignment model is proposed in [Alkhouli & Ney 17] (Section 4.5.7). A variant based on self-attentive layers is presented in [Alkhouli & Bretschner++ 18] (Section 4.7.3). [Schwenk 12] propose a feedforward network that
4.3 Terminology

A neural network can be represented by a weighted directed graph. The graph consists of nodes and edges, where the edges have weights that are called the network parameters. We are interested in specific structures that have nodes arranged in layers. Each node has an output called activation. The activation is computed using an activation function. The first layer in the network is the input layer, the intermediate layers are hidden layers, and the last layer is called the output layer. We distinguish between two main types of layers. A feedforward layer has nodes that only receive input from other layers. These layers are acyclic. A recurrent layer has nodes that can receive input from the layer itself, hence, they are cyclic. When the network receives new input, we mark this as a time step. In text processing, a time step corresponds to a word position in the sequence; therefore, whenever we refer to time in this work, we refer to a certain word position in the corpus. Recurrent layers maintain states over time. A state represents activation values computed for a given time step. The state at a previous time step is used together with the recurrent layer input to compute the layer output and the next state. Feedforward layers, on the other hand, are stateless. They compute their output solely using the layer input, without requiring layer values at a previous time step. Neural networks that only have feedforward layers compute phrase scores offline. The neural scores are used to score phrases in a phrase-based system.

[Le & Allauzen+ 12] present translation models using an output layer with classes and a short-list for rescoring using feedforward networks. They compare between word-factored and tuple-factored n-gram models, obtaining their best results using the word-factored approach, which is less amenable to data sparsity issues. [Kalchbrenner & Blunsom 13] use recurrent neural networks with full source sentence representations. The continuous representations are obtained by applying a sequence of convolutions on the source sentence, and the result is fed into the hidden layer of a recurrent language model, making it conditioned on the source sentence. Rescoring results indicate no improvements over the state of the art. [Auli & Galley+ 13] also include source sentence representations built either using Latent Semantic Analysis or by concatenating word embeddings. They report no notable gain over systems using a recurrent language model.

Recurrent language models are first proposed in [Mikolov & Karafiát+ 10]. [Sundermeyer & Schlüter+ 12] propose using LSTM layers as recurrent layers for language modeling. The authors use class-factored output layers. The model is described in Section 4.5.4.

The recurrent encoder-decoder model described in Section 4.6.1 is proposed by [Cho & van Merriënboer+ 14b, Sutskever & Vinyals+ 14]. The model is later extended with the attention component in [Bahdanau & Cho+ 14]. In Section 4.6.2, we describe an attention model variant close to the model proposed in [Luong & Pham+ 15]. [Alkhouli & Ney 17] propose to bias the attention energies using external alignment information (Section 4.7.1). [Vaswani & Shazeer+ 17] propose replacing recurrent layers with self-attentive layers of multiple attention heads in the encoder and decoder layers. The work also proposes using multi-head attention between the encoder and decoder layers. The model is described in Section 4.6.3. [Alkhouli & Bretschner+ 18] propose adding extra alignment heads to the source-target multi-head attention components to introduce explicit alignment to the model (Section 4.7.2).

While our focus in this work is on the task of machine translation itself, it is worth noting that translation can be used to model contextual word representations, which can be useful for various NLP tasks. For instance, [Rios & Aziz+ 18] propose a word representation model that marginalizes over a latent word alignment between source and target sentences. It uses a uniform alignment model, similar to IBM 1, however, the model is parameterized by a neural network instead of categorical parameters.
are called \textit{feedforward networks}, and networks that have at least one recurrent layer are called \textit{recurrent networks}.

### 4.4 Neural Network Layers

Neural networks usually have a multi-layer structure including input, hidden and output layers. We focus on layers that are of interest to language modeling and machine translation. These networks receive text as input and output. Below is a description of some of the different layers that are used to construct such neural networks.

#### 4.4.1 Input Layer

The neural networks we use receive words as input. The words are fed into the network in the form of vectors. Formally, let $V_e$ define the set of vocabulary words of size $|V_e|$, and let the word $e \in V_e$ be the $k$-th word in the vocabulary. The word $e$ is represented by a one-hot vector $\hat{e} \in \{0, 1\}^{|V_e|}$ defined as follows:

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

where $m$ denotes the $m$-th position in the vector.

#### 4.4.2 Embedding Layer

The embedding layer is a hidden layer that maps the sparse one-hot vector representation into a real-valued continuous-space representation, commonly referred to as the word embedding vector. The layer has the parameter weights $A_1 \in \mathbb{R}^{E \times |V_e|}$, where $E$ is the embedding vector dimension. The continuous word representation is computed by multiplying the weight matrix with the one-hot representation, which amounts to selecting a column of the weight matrix as follows:

$$\tilde{e} = A_1 \hat{e}.$$  

The matrix $A_1$ is usually referred to as the word embedding matrix, since each column corresponds to one vocabulary word, embedding it in the continuous vector space. $\tilde{e}$ is an $E$-dimensional word vector (embedding) of the word $e$.

$n$-gram feedforward networks expect $n - 1$ words as input. The embedding layer looks up the word embeddings and concatenates them. Given the $(n - 1)$-gram $e_{i-n+1}, e_{i-n+2}, \ldots, e_{i-1}$, the output of the embedding layer is

$$y^{(1)} = A_1 \hat{e}_{i-n+1} \circ A_1 \hat{e}_{i-n+2} \circ \ldots \circ A_1 \hat{e}_{i-1},$$

where $\circ$ defines the concatenation operation. $y^{(1)}$ denotes the output of the embedding layer, considered to be the first hidden layer. From now on we will use the superscript notation in parentheses $y^{(l)}$ to denote the $l$-th hidden layer in the network.

#### 4.4.3 Feedforward Layer

The feedforward layer is composed of an affine transformation of the input followed by an activation function. The $l$-th feedforward layer receives input from the previous layer $l - 1$, but it might as well receive input from other another layer which needs to be computed first. Let
4.4 Neural Network Layers

Let \( y^{(l-1)} \in \mathbb{R}^{S_{l-1} \times 1} \) be the output of the \((l-1)\)-th layer, the output of the feedforward layer at depth \( l \) is given by

\[
y^{(l)} = f(A_l y^{(l-1)} + b_l),
\]

where \( A_l \in \mathbb{R}^{S_l \times S_{l-1}} \) is the weight matrix and \( b_l \in \mathbb{R}^{S_l \times 1} \) is the bias associated with the layer. These are free parameters estimated during training. \( f \) is an element-wise non-linear activation function. Examples of such activation function include the hyperbolic tangent, which is defined as

\[
tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}. \tag{4.1}
\]

It is important to use a function that is differentiable almost everywhere to train the parameters using gradient-based approaches. The derivative of \( \tanh \) is given by

\[
tanh'(u) = 1 - \tanh^2(u). \tag{4.2}
\]

Another common non-linear function is the sigmoid

\[
\sigma(u) = \frac{1}{1 + e^{-u}}. \tag{4.3}
\]

Its derivative is given by

\[
\sigma'(u) = \sigma(u)(1 - \sigma(u)). \tag{4.4}
\]

4.4.4 Recurrent Layer

Recurrent layers are different to feedforward layers in that they have a state maintained over time. Computing the output of a recurrent layer includes input from the current time step in addition to the output of the layer itself at the previous time step; therefore, it is important for efficient computation to maintain the layer output at the previous time step to compute the current output. If the output of the previous layer at the current time step \( i \) is \( y^{(l-1)}_i \in \mathbb{R}^{S_{l-1} \times 1} \), the recurrent layer output at time \( i \) is given by

\[
y^{(l)}_i = g(A_l y^{(l-1)}_i + b_l),
\]

where \( A_l \in \mathbb{R}^{S_l \times S_{l-1}} \) and \( B_l \in \mathbb{R}^{S_l \times S_l} \) are weight matrices and \( b_l \in \mathbb{R}^{S_l \times 1} \) is a bias vector, to be estimated during training. \( g \) is an element-wise non-linear activation function.

Recurrent layers have a fundamental implication for sequence modeling. While the recurrent term is directly dependent on the previous time step only, it introduces implicit full dependence on all previous time steps, making it dependent on the full context. This can be shown by unfolding the recurrency as follows:

\[
y_1^{(l)} \rightarrow y_2^{(l)} \rightarrow \ldots \rightarrow y_{i-1}^{(l)} \rightarrow y_i^{(l)} \rightarrow \ldots \uparrow \rightarrow \ldots \rightarrow y_i^{(l-1)} \rightarrow \ldots,
\]
where arrows indicate direct dependence between layers. In contrast, feedforward layers are limited to the input context. This makes recurrent layers appealing for sequence modeling tasks in general, and for natural language processing tasks in particular, where decisions are affected by long-range dependencies. Feedforward networks that merely consist of feedforward layers can only capture limited context by design.

4.4.5 Long Short-Term Memory Layer

In principle, standard recurrent layers should be able to capture unbounded long-range context. However, it is found out in practice that training networks that have such layers is problematic. This is because training involves computing the gradient, which becomes increasingly unstable as the sequence length gets longer [Bengio & Simard+ 94, Hochreiter & Bengio+ 01]. The long context dependence can result in exponentially exploding gradients in the case of multiplying error values greater than 1, which affects convergence and leads to parameter oscillation. When the error values are less than 1, multiplication over a long sequence results in exponentially decaying gradients. When the gradient values are close to zero, there is effectively no learning to take place. These issues are commonly referred to as the exploding and vanishing gradient problems. Note that these problems can also arise in deep feedforward networks that stack a large number of layers for the same reasons.

There has been several proposals in the literature to mitigate these problems. One commonly used solution is the long short-term memory (LSTM) layer [Hochreiter & Schmidhuber 97]. The idea is to control the error flow by introducing a type of unit called constant error carousel (CEC), which has a scaling factor fixed to 1 for gradients passing through it. The CEC uses a linear activation function and a recursive weight of 1. Due to its limited capability, it is further extended by input and output gating functions that produce values in the (0, 1) interval using the sigmoid function. They dynamically control the input and output that flows through the CEC unit. The LSTM architecture is additionally extended by forget gates to reset the CEC [Gers & Schmidhuber+ 00], and by peephole connections [Gers & Schraudolph+ 02]. In this dissertation we use the refined version of the LSTM that has all these extensions.

We use single-cell LSTM layers. If the input of the LSTM layer is \( x_i \in \mathbb{R}^{S_{i-1} \times 1} \), the LSTM equations are given by

\[
\begin{align*}
n_i &= \tanh(Ax_i + Bh_{i-1}) \\
m_i &= \sigma(A_m x_i + B_m h_{i-1} + W_m \odot c_{i-1}) \\
f_i &= \sigma(A_f x_i + B_f h_{i-1} + W_f \odot c_{i-1}) \\
c_i &= f_i \odot c_{i-1} + m_i \odot n_i \\
o_i &= \sigma(A_o x_i + B_o h_{i-1} + W_o \odot c_{i-1}) \\
h_i &= o_i \odot \tanh(c_i),
\end{align*}
\]

where \( \odot \) denotes element-wise multiplication, \( h_i = y_i^{(l)} \in \mathbb{R}^{S_i \times 1} \) is the layer output at step \( i \). The layer weights are the matrices \( A, A_m, A_f, A_o \in \mathbb{R}^{S_i \times S_i-1} \), the recurrent weights \( B, B_m, B_f, B_o \in \mathbb{R}^{S_{i-1} \times S_{i-1}} \), and the peephole connection weight vectors \( W_m, W_f, W_o \in \mathbb{R}^{S_i \times 1} \). The equations do not include bias vectors for simplicity but they can be added. \( n_i \) computes an intermediate representation of the input, \( m_i \) is the input gate, \( f_i \) is the forget gate, and \( o_i \) is the output gate. A gate computes values in the (0, 1) interval acting like a soft switch. In the extreme case of zero forget gate values, the cell state is reset by ignoring the previous state \( c_{i-1} \), and the new state \( c_i \) is computed using the input \( n_i \), which is gated by \( m_i \). The output values \( h_i \) are also gated using the output gate \( o_i \).

Recently, [Cho & van Merrienboer+ 14b] introduced a variant close to LSTMs called gated recurrent units (GRUs). It is similar to the LSTM in that it makes use of gating, but it uses less
4.4 Neural Network Layers

number of parameters. Experiments in the literature indicate no difference [Chung & Gülçehre+ 14] or better performance for LSTM layers [Irie & Tüske+ 16]. In this work, we only use LSTM layers.

### 4.4.6 Softmax Layer

The softmax layer computes a normalized probability distribution of a discrete random variable. The number of nodes in the layer is determined by the number of values the random variable can assume. In natural language processing, a common random variable is the word \( e \in V_e \), which can assume \(|V_e|\) many values. In this case, the softmax layer is of size \(|V_e|\) nodes. If the input to the softmax layer is \( x \in \mathbb{R}^{|V_e| \times 1} \), the softmax output at index \( j \in \{1, 2, ..., |V_e|\} \) is given by

\[
\text{softmax}(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{|V_e|} e^{x_k}}, \tag{4.5}
\]

where \( x_j \) and \( x_k \) are respectively the values at indices \( j \) and \( k \) of the input vector \( x \). The function ensures all values are positive using the exponent function, and normalizes the scores using the normalization factor in the denominator. This results in a probability distribution.

### 4.4.7 Output Layer

The output layer computes a probability distribution over the target vocabulary \( V_e \). If the input vector to the layer is given by \( y_{i}^{(L-1)} \in \mathbb{R}^{S_{L-1} \times 1} \), the output layer is computed as

\[
y_{i}^{(L)} = \text{softmax}(A_{L-1}y_{i}^{(L-1)}),
\]

where \( A_{L-1} \in \mathbb{R}^{|V_e| \times S_{L-1}} \) is the output embedding matrix.

Computing the output layer for large vocabularies using a large output embedding matrix is costly. This is due to the normalization sum in the denominator of Equation 4.5, which requires computing the full output vector. Several solutions have been proposed to reduce the computational cost of this softmax computation, including noise contrastive estimation [Gutmann & Hyvärinen 10, Mnih & Teh 12, Vaswani & Zhao+ 13], subword units [Sennrich & Haddow+ 16], and the hierarchical softmax [Morin & Bengio 05].

We use a shallow hierarchical softmax output layer in some of our experiments in this dissertation. This corresponds to a hierarchical softmax of depth 1 [Mikolov & Kombrink+ 11]. In this case, the output layer is factored into two layers: a word class layer and a word layer conditioned on the class. Given a fixed class set \( \mathcal{C} \) and a mapping function \( c(e) \) given by

\[
c : V_e \rightarrow \mathcal{C} \quad \quad e \mapsto c(e),
\]

which maps words into classes, where each word is mapped to one and only one class, the class-factored output layer is given by

\[
p(e_i | c(e_i)^{-1}, f_i^L) = p(c(e_i) | e_i^{-1}, f_i^L) \cdot p(e_i | c(e_i), e_i^{-1}, f_i^L). \tag{4.6}
\]

The class-factored output layer is illustrated in Figure 4.1. The class and word probability distributions are computed using softmax output layers. When \(|V_e| \gg |\mathcal{C}|\), computing the class
Figure 4.1: A class-factored output layer composed of a class layer (top left) and a word layer (top right).

probability is computationally cheap. The word probability computation only involves normalization over the words in the class \( c(e_i) \), i.e. \( w \in V_e \) s.t. \( c(w) = c(e_i) \), since \( p(e_i|g, e_{i-1}, f_1^t) = 0 \) if \( g \neq c(e_i) \). This constrains the softmax computation to a subset of the vocabulary, reducing its computational cost. Consider a class mapping where each class has \( |V_e|/|C| \) words, the time complexity of computing the class-factored output layer is \( O(|C| + |V_e|/|C|) \), which has a minimum at \( |C| = \sqrt{|V_e|} \). This results in a complexity of \( O(\sqrt{|V_e|}) \) for computing the class-factored output layer. In comparison, using a full output layer has a time complexity of \( O(|V_e|) \). Hence, using a class-factored output layer can heavily reduce the computation cost, depending on the class mapping.

There are different approaches to define the class mapping. [Mikolov & Kombrink+ 11] proposed the use of frequency binning to generate classes equal in size. [Shi & Zhang+ 13, Zweig & Makarychev 13] compared that to perplexity-based word clustering, and found the latter to perform better in terms of perplexity. We use perplexity-based clustering in this work trained using the so-called exchange algorithm [Kneser & Ney 91].

4.5 Neural Network Models For Machine Translation

Having introduced the basic components used to construct neural networks, we will now describe the full architecture of lexicon and alignment neural network models used in this thesis. The lexicon model takes source and target words as input, and optionally the word alignment between them. It computes a probability distribution over the target words. In contrast, the alignment model computes a probability distribution over relative alignment jumps. The lexical and alignment models described here are alignment-based models. We use this terminology to highlight that these models get the alignment path as external input to the models. That is, they are conditioned on the alignment path. In contrast, attention-based models have integrated components that dynamically compute attention weights as part of the model instead. Attention weights can be used to extract word alignment. Attention-based models do not use external alignment information as input. We discuss attention-based models in Section 4.6. In the following we will describe alignment-based feedforward and recurrent neural network model architectures.

4.5.1 Feedforward Lexicon Model

The feedforward lexicon model proposed in [Devlin & Zbib+ 14] defines a fixed-sized source window of size \( 2m + 1 \) used as input to the network. It also uses a fixed-sized target window of \( n \) words to include the target history. The model receives the aligned source position \( b_i \) as input. The source window is centered around the source word \( f_{b_i} \) at source position \( b_i \). It captures previous and future words surrounding the source word \( f_{b_i} \) to predict target word \( e_i \). The model
4.5 Neural Network Models For Machine Translation

defines a shared source embedding matrix $F$ and a shared target embedding matrix $E$. The embeddings of the source and target words are looked up and concatenated as follows:

$$
F \hat{f}_{b_i - m} \circ F \hat{f}_{b_i - m+1} \circ \ldots \circ F \hat{f}_{b_i + m-1} \circ F \hat{f}_{b_i + m} \circ E \hat{e}_{i-n} \circ \ldots \circ E \hat{e}_{i-1},
$$

where $\hat{f}$ and $\hat{e}$ denote the one-hot vector representation. The concatenation result is passed to the rest of the network. The model output over the target vocabulary is then conditioned as follows:

$$
p(e_i | e_{i-1}, f_{b_i - m}).
$$

Note that the model does not depend on the full alignment path, rather, it only requires the current alignment point, making it a zero-order model with respect to the alignment. The model computes the probability of a target sequence as a product over all target positions

$$
p(e_1 | f_1, b_1) = \prod_{i=1}^{I} p(e_i | e_{i-n}, f_{b_i - m}).
$$

An example of such model is illustrated in Figure 4.2 where $m = 2$ and $n = 3$. The model includes 5 source words and 3 target words. The output layer uses the class factorization described in Section 4.4.7.

4.5.2 Feedforward Alignment Model

The alignment model computes a probability distribution over relative alignment jumps $\Delta_i = b_i - b_{i-1}$, where $b_{i-1}$ is the last aligned source position and $b_i$ is the current source po-

Figure 4.2: An example of a feedforward lexicon model, with a 3-gram target history and a source window of 5 words.
sition to be predicted. We choose to model jumps for two reasons. First, modeling the absolute position requires defining a maximum source sequence length, while modeling the relative jump is more flexible, as it only requires defining a maximum for the relative jump, allowing to handle source sequences of arbitrary length. Second, modeling the relative jump allows sharing training observations that have equal jumps. E.g., if we have a monotone alignment \( b_i = i \) between the target sequence \( e^1_i \) and the source sequence \( f^1_i \) where \( I = J \), the training sample will only have the relative jump \( \Delta_i = 1 \) for all positions. In contrast, modeling the absolute jumps trains sparse \( I \) many alignment predictions. We assume that training with densely observed events generalizes better than training with sparsely observed events.

The feedforward alignment model first described in [Alkhouli & Bretschner + 16] uses an architecture similar to the feedforward lexicon model. There are two main differences to the lexicon model. First, the source window is centered at the source position \( b_{i-1} \) instead of \( b_i \). This is because at time step \( i \) the model predicts the jump to \( b_i \), and therefore it cannot be used as input. This means that the model predicts the jump from \( b_{i-1} \) to \( b_i \) using the source context surrounding the source word aligned in the previous step. The alignment model uses the same target history as the lexicon model. This results in a first-order model with respect to the alignment. The embeddings of the source and target words are looked up and concatenated as follows:

\[
F \hat{f}_{b_{i-1}-m} \circ F \hat{f}_{b_{i-1}-m+1} \circ \cdots \circ F \hat{f}_{b_{i-1}+m} \circ E \hat{e}_{i-n} \circ \cdots \circ E \hat{e}_{i-1},
\]

where \( \hat{f} \) and \( \hat{e} \) denote the one-hot vector representation. The concatenation result is passed to the rest of the network. The second difference to the lexicon model is that the alignment model computes a probability distribution over the source jumps instead of the target words. The model output is then conditioned as follows:

\[
p(\Delta_i | e^1_{i-n}, f^1_{b_{i-1}-m}).
\]

The score of the full alignment path is given by

\[
p(b^1_i | e^1_1, f^1_1) = \prod_{i=1}^I p(\Delta_i | e^1_{i-n}, f^1_{b_{i-1}-m}).
\]

An example of a feedforward alignment model is shown in Figure 4.3, with \( m = 2 \), and \( n = 3 \). Note that the model does not use a class-factored output layer because the output layer size is much smaller compared to the lexicon output layer, making it much cheaper to compute without any workarounds.

### 4.5.3 Recurrent Neural Network Models

The feedforward neural network models presented so far have limited expressive power. This is because they are conditioned on limited context. In sequence-to-sequence problems like machine translation, this is an unnecessary restriction. Long-range dependencies between words in the sequence cannot be captured without having large source and target windows. Although this is possible in principle, it is prohibitive in practice, since the size of the embedding layer increases linearly with respect to \( m \) and \( n \), which respectively determine the size of the source and target contexts. This increases the model parameters, and the memory and time requirements. One way of making feedforward networks capture longer context is by using convolutional layers [LeCun & Boser + 89]. Another approach that is appealing for sequence-to-sequence modeling is using recurrent networks. In this work, we focus on the recurrent network approach since it dominates the field of natural language processing while demonstrating superior results. A third approach is to use self-attention layers which are part of the state-of-the-art so-called transformer models [Vaswani & Shazeer + 17]. We will discuss self-attention models in Section 4.6.3.
4.5 Neural Network Models For Machine Translation

Figure 4.3: An example of a feedforward alignment model [Alkhouli & Bretschner+ 16], with a 3-gram target history and a source window of size 5 words.

4.5.4 Recurrent Neural Network Language Models

Recurrent language models [Mikolov & Karafiát+ 10] use at least one recurrent hidden layer as illustrated in Figure 4.4. The recurrency leads to a layer state that is dependent on all previous input. To predict word $e_i$, only the word $e_{i-1}$ is received as immediate input. The network maintains a state associated with the recurrent layer that is dependent on the full previous input $e_{i-2}$. Both the hidden state and the input are used to compute the network output. Hence, the output is conditioned on the full context $e_{i-1}$. The recurrent network can be represented as a feedforward network unfolded over time, as shown in Figure 4.5. Because the feedforward network is conditioned on its input only, modeling longer context requires additional network input at each step. While recurrent networks are able to model long context with limited input per time step, there is an overhead due to the recurrent states that need to be stored if the network is to be evaluated efficiently.

The detailed equations corresponding to model in Figure 4.4 including class-factorization are

\[
\begin{align*}
y_{i-1} &= A_1 \hat{e}_{i-1} \\
h(e_{i-1}^1) &= \xi(y_{i-1}; A_2, h(e_{i-2}^1)) \\
o(e_{i-1}^1) &= A_3 h(e_{i-1}^1) \\
o(c(e_i))(e_{i-1}^1) &= A_{c(e_i)} h(e_{i-1}^1) \\
p(c(e_i)|e_{i-1}^1) &= \text{softmax}(o(c(e_i))(e_{i-1}^1))|_{c(e_i)} \\
p(e_i|c(e_i), e_{i-1}^1) &= \text{softmax}(o(c(e_i))(e_{i-1}^1))|_{e_i},
\end{align*}
\]

where $A_1, A_2, A_3$ denote the neural network weight matrices, $A_{c(e_i)}$ denote the class-specific network matrices indexed by the class $c(e_i)$. Computing the probability of $e_i$ only requires the
use of the matrix $A_{c(e_i)}$, which is a subset of the full word output matrix. $\hat{e}_{i-1}$ is the one-hot vector encoding the word $e_{i-1}$, and $y_{i-1}$ is its word embedding vector. $h$ is a vector of the hidden layer activations depending on the unbounded context, and it is computed recurrently using the function $\xi$, which we use to represent an abstract recurrent layer. $h(e_{i-1})$ is the hidden state of the layer. $o(\cdot) \in \mathbb{R}^{|C|}$ is a $|C|$-dimensional vector containing the raw unnormalized class layer values, where $|C|$ is the number of classes. $o_{c(e_i)}(\cdot) \in \mathbb{R}^s$ is an $s$-dimensional vector of unnormalized word layer values, where $s = \left| \{ e \in V_e \text{ s.t. } c(e) = c(e_i) \} \right|$ is the size of the class $c(e_i)$.

### 4.5.5 Recurrent Neural Network Lexical Model

We can convert a language model into a translation model by conditioning it on (part of) the source sentence. Figure 4.6 shows an example of a translation lexical model. In comparison to the language model in Figure 4.4, the model receives the source word $f_{b_i}$ at source position $b_i$ as extra input at each target step $i$. This assumes that the alignment between source and target words is known. Due to the recurrency, the network output at step $i$ is conditioned on
4.5 Neural Network Models For Machine Translation

Figure 4.6: An example of a recurrent neural network translation model using a class-factored output layer. The model captures the full target history $e_{i-1}$, and the partial source context $[f_{b_i}]_{i=1}^i = f_{b_1}, f_{b_2}, ..., f_{b_i}$ through the hidden recurrent layer, denoted with a curved arrow. At each target step $i$, the target input word $e_{i-1}$ and the source word $f_{b_i}$ are received as input. Note that the source word is the word aligned to the target word to be predicted at the current target step $i$. If the dashed part is dropped, a translation model without any dependence on the target side is obtained.

We can have another variant that does not depend on the target words by dropping the dashed part in Figure 4.6. This embedding layer is then given by

$$y_{i-1} = A_t e_{i-1} + A_s f_{b_i},$$

where $f_{b_i}$ is the one-hot representation of the source word at source position $b_i$ aligned to the target word at position $i$, which is to be predicted. $A_t$ and $A_s$ are the target and source word embedding matrices. Here, the source and target embeddings are looked up and aggregated. The source and target embeddings are assumed to have equal dimensions. The rest of the network can be computed using Equations 4.8 – 4.12, with the new source dependence $[f_{b_i}]_{i=1}^i$ in addition to the original dependence on the target history $e_{i-1}$. The output of the network is

$$p(e_i | [e_{i-1}, f_{b_i}]) = p(c(e_i) | [e_{i-1}, f_{b_i}]_{i=1}^i) \cdot p(e_i | c(e_i), [e_{i-1}, f_{b_i}])_{i=1}^i).$$

Both variants including and excluding the target words were proposed in [Sundermeyer & Alkhouli+ 14], where the authors apply the models to modified source and target sequences of equal length, i.e. $I = J$. The modified sequences are obtained by applying heuristics to insert special tokens for unaligned and multiple-aligned words. Note that these models do not leverage the full source information, which we assume to be given at any time step. These models are called unidirectional models, since they have recurrent layers that process information in one direction.
of time (from $i - 1$ to $i$). They are distinguished from bidirectional networks, which include layers that run in both forward and backward time directions. We introduce bidirectional lexical models next.

### 4.5.6 Bidirectional Recurrent Neural Network Lexical Model

One shortcoming of unidirectional models is that they do not capture the full source context, which is available information that can be exploited. In speech recognition, bidirectional models were first proposed by [Schuster & Paliwal 97] to capture the full input. In machine translation, alignment-based bidirectional models were first proposed by [Sundermeyer & Alkhouli 14]. Where they were applied within the phrase-based framework. Bidirectional models were independently proposed around the same time by [Bahdanau & Cho 14]. The latter models were used for standalone neural machine translation. [Alkhouli & Ney 17] apply alignment-based bidirectional models to perform neural machine translation. In the following we discuss alignment-based bidirectional neural models for machine translation.

An extra recurrent layer can be added to include the additional source information not included in the unidirectional model. This layer processes information in the backward direction of time (from $i + 1$ to $i$), such that the information $f_{b_i}^+$ is processed after $f_{b_i + 1}^+$. This is justified as follows: since the target word $e_i$ to be predicted is aligned to the source word $f_{b_i}$, the intuition is that the target word is a translation of this source word. Hence, we compute the recurrent state such that the most relevant information is processed at the last step, so that the state will be mostly influenced by the most recent relevant input. This assumes that recurrent states will be more influenced by the most recent input and less influenced by input that has been proceed in a distant time step. An example of a bidirectional lexical model is illustrated in Figure 4.7. This example shows the network unfolded over time. The additional recurrent backward layer is marked in yellow. The example uses an additional forward recurrent layer before the output layer to mix the forward and backward information it receives from the forward and backward recurrent layers at the bottom. The example uses a full word output layer for simplicity, although a class-factored output layer can be used as well. The network can be described using the following equations:

\[
x_i = A_s \hat{f}_i \\
y_{i - 1} = A_t \hat{e}_{i - 1} + x_i \\
h_i^+ = \xi(y_{i - 1}; A_2, h_{i - 1}^-) \\
h_i^- = \xi(x_i; A_3, h_{i + 1}^-) \\
m_i = A_4 h_i^+ + A_5 h_i^- \\
g_i = \xi(m_i; A_6, g_{i - 1}) \\
o_i = A_7 g_i \\
p(e_i|e_{i - 1}, f_l^I, b_l^I) = \text{softmax}(o_i)|_{e_i}
\]

where $A_s$, and $A_t$ are respectively the source and target word embedding matrices, and $A_2, \ldots, A_7$ are weight matrices. $\xi$ denotes an abstract recurrent layer function. To highlight that the source and target words are paired at the embedding level, we use the following notation:

\[
p(e_i|e_{i - 1}, f_l^I, b_l^I) = p(e_i|e_{i - 1}, f_{b_i}^I|_{i = 1}, [f_{b_i}^I]_{i = 1}^I),
\]

where $[e_{i - 1}, f_{b_i}^I]_{i = 1}^I$ highlights that the pairing is done on the word level, and $[f_{b_i}^I]_{i = 1}^I$ denotes the remaining source part captured by the backward layer in Equation 4.16.
A variant of the bidirectional model that excludes the target input is obtained by modifying Equation 4.14 to

\[ y_{i-1} = x_i, \]
while keeping the other equations unchanged. This corresponds to dropping the solid red components in Figure 4.7. The model output is denoted by
\[
p(e_i|f^I_i, b^I_i) = p(e_i|f^I_i, b^I_i = 1, b^I_i = 1).
\] (4.21)

Note that the bidirectional lexical models discussed above require knowledge of the full alignment path \(b^I_i\) at all time steps, which is needed to determine the order in which the source words are fed into the network. The backward layer is dependent on alignment information that is not available during decoding; therefore, these bidirectional variants cannot be used during decoding. [Sundermeyer & Alkhouli + 14] propose to use these models in offline \(N\)-best rescoring setups, where the complete alignment path is already generated during first-pass decoding.

[Alkhouli & Rietig + 15] propose another variant that computes the bidirectional source states on the source sentence independent of the alignment. A description of such a model is given by the following equations:
\[
x_j = A_s \hat{f}_j
\] (4.22)

\[
h_j^+ = \xi(x_j; A_2, h_{j-1}^+)
\] (4.23)

\[
h_j^- = \xi(x_j; A_3, h_{j+1}^-)
\] (4.24)

\[
m_j = A_4 h_j^+ + A_5 h_j^-
\] (4.25)

\[
g_j = \xi(m_j; A_6, g_{j-1})
\] (4.26)

\[
o_i = A_7 g_i
\] (4.27)

\[
p(e_i|f^I_i, b_i) = \text{softmax}(o_i) |_{e_i}.
\] (4.28)

In this variant, the mapping \(b_i\) of the target position to the source position is used in Equation 4.27. This makes the model conditioned on the alignment point \(b_i\) only, and not on the full alignment path, since there are no recurrent layers that depend on \(b_i\). Note that this model is independent of the target history; therefore, it can be thought of as a bag of words model with respect to the target sequence, where the score is independent of the target sequence order. In this dissertation, we rely on decoding algorithms that are monotonic in the target side, that is, they generate the target sequence in order from left to right. This variant can be used in such decoding algorithms.

A more expressive variant is obtained by computing additional recurrent states on the target sequence, which makes the model conditioned on the full target history \(e_{i-1}^T\). In addition, the alignment history \(b_{i-1}^I\) can be included by adding the alignment information before a recurrent layer. An example of such a network is shown in Figure 4.8. Using this architecture, the alignment path for the full target word sequence is no longer needed. This model requires the alignment path up to and including the current target position \(b^I_i\). The model is described using the following equations:
\[
x_j = A_s \hat{f}_j
\] (4.29)

\[
y_{i-1} = A_t \hat{e}_{i-1}
\] (4.30)

\[
t_{i-1} = \xi(y_{i-1}; A_2, t_{i-2})
\] (4.31)

\[
h_j^+ = \xi(x_j; A_3, h_{j-1}^+)
\] (4.32)

\[
h_j^- = \xi(x_j; A_4, h_{j+1}^-)
\] (4.33)

\[
h_j = A_5 h_j^+ + A_6 h_j^-
\] (4.34)

\[
m_i = A_7 t_{i-1} + h_i
\] (4.35)

\[
g_i = \xi(m_i; A_8, g_{i-1})
\] (4.36)

\[
o_i = A_9 g_i
\] (4.37)

\[
p(e_i|e_{i-1}^T, f^I_i, b^I_i) = \text{softmax}(o_i) |_{e_i},
\] (4.38)
4.5 Neural Network Models For Machine Translation

\[ p(e_i | e_i^{-1}, f_i^j, b_i^j) \]

Figure 4.8: An example of the bidirectional lexical model with paired source and target hidden states. The yellow color is used for source information, and red is used for target information. The gradient color indicates mixed source and target information. At time step \( i \) the target word \( e_i \) is to be predicted. The full source information \( f_1, f_2, ..., f_J \) is assumed to be given at any time step. The partial alignment path until the current step \( b_i^1 \) is also assumed given. A forward recurrent layer (from \( j - 1 \) to \( j \) and a backward recurrent layer (from \( j + 1 \) to \( j \)) are computed for the source sequence. The aggregation of the forward and backward source layers (both in yellow) at source position \( j \) is used as a representation for the source information. The source representation is denoted \( h_j \). Another independent forward recurrent layer (red) is computed on the target sequence. The source and target states are aggregated and fed to a recurrent layer (gradient color) that mixes the source and target information. Note that the pairing between the source and target is done between the hidden states instead of the embeddings. The source representation \( h_j \) to be paired with the target representation is selected using the alignment point \( j = b_i \), that is, \( j \) is the source position aligned to the target word \( e_i \) to be predicted at the output layer. Because the source representation is computed on the original word order of the source sentence, the model only requires the alignment path \( b_i^1 \) up to and including the alignment of the current step. The alignment of the future unknown target words is not required. Note the difference to the model in Figure 4.7 that requires the full alignment information. Again, we can obtain a variant that does not include the target input by removing the target word embeddings and the target recurrent layer (red components). In this case the model is only conditioned on the source and alignment information \( p(e_i | f_i^j, b_i^j) \).

where \( A_s \) and \( A_t \) are the source and target word embedding matrices, \( A_2, ..., A_9 \) are weight matrices.
Due to the recurrent layer in Equation 4.36, the model is dependent on the complete past alignment path $b^1_i$, making it a high-order model with respect to the alignment variable. If this layer is removed, the model becomes conditioned on the current alignment point $b_i$ only, where the posterior distribution is given by $p(e_i|e_{i-1}^1, f_i^1, b_i)$. This distinction is important to consider if the forward-backward algorithm is to be used to train such a model as described in [Wang & Alkhouli++ 17]. The complexity of the forward-backward algorithm for $n$-order models is $O(J \cdot J^{n+1})$. If the model is of high order, the forward-backward algorithm cannot be computed efficiently. In [Wang & Alkhouli++ 17, Wang & Zhu++ 18] the authors train first-order models using the forward-backward algorithm. In this dissertation, we do not use the forward-backward algorithm to train the neural models, since they are high-order models.

4.5.7 Bidirectional Recurrent Neural Network Alignment Model

The state-paired bidirectional lexical model architecture can be used to derive a bidirectional alignment model. This is similar in concept to the feedforward alignment model proposed in Section 4.5.2, where the model predicts the source jump $\Delta_i = b_i - b_{i-1}$. Unlike the feedforward alignment model, the bidirectional alignment model leverages the full source context and the complete target history to make predictions. The model is proposed in [Alkhouli & Ney 17]. An example of the bidirectional alignment model is shown in Figure 4.9. Since the model predicts the current alignment position $b_i$, it cannot be included as input; therefore, the state-pairing between the source representation $h_j$ and the target representation is based on the predecessor alignment point $b_{i-1}$. This is different from the lexical model that pairs states using the alignment point $b_i$.

The model is obtained by changing Equation 4.35 to

$$m_i = A \Delta_{t_{i-1}} + h_{b_{i-1}}.$$ 

The model is conditioned on the alignment path $b_{i-1}^1$

$$p(\Delta_i|e_{i-1}^1, f_i^1, b_{i-1}^1),$$

which has a high-order with respect to the alignment variable. A first-order model is obtained by removing the recurrent layer of Equation 4.36.

4.6 Attention-Based Neural Network Models

The lexical and alignment models discussed so far are alignment-based, that is, they depend on external alignment information. There is another family of neural models that do not depend on the word alignment. These models are appealing because they are trained end-to-end, without intermediate training steps that require extra information not generated by the model itself. These models are based on the encoder-decoder architecture described in Section 4.6.1. Afterwards, we describe attention-based recurrent neural models in Section 4.6.2. Most recently, transformer models were proposed to replace attention-based recurrent models. These models replace recurrent layers with feedforward layers and use multiple attention components. We describe transformer models in Section 4.6.3.

4.6.1 Encoder-Decoder Recurrent Neural Network

As suggested by the name, encoder-decoder neural networks are composed of two main components: the encoder and the decoder [Cho & van Merrienboer++ 14b, Sutskever & Vinyals++ 14]. The encoder part encodes the source sequence into a fixed-sized continuous vector representation,
4.6 Attention-Based Neural Network Models

\[ p(\Delta_i | e_{i-1}, j, f_1, b_i^{-1}) \]

Figure 4.9: An example of the bidirectional alignment model. The yellow color is used for source information, and red is used for target information. The gradient color indicates mixed source and target information. At time step \( i \) the alignment jump \( \Delta_i = b_i - b_{i-1} \) from the last aligned source position \( b_{i-1} \) to the currently aligned source position \( b_i \) is predicted. The full source information \( f_1, f_2, ..., f_J \) is assumed to be given at any time step. The partial alignment path until the last step \( b_{i-1} \) is also assumed given. The source and target states are aggregated and fed to a recurrent layer (gradient color) that mixes the source and target information. The source representation \( h_j \) to be paired with the target representation is selected using the last alignment point \( j = b_{i-1} \), that is, \( j \) is the source position aligned to the previous target word \( e_{i-1} \). The current alignment point \( b_i \) is to be predicted, and therefore it cannot be used as input to the model. Note the difference to the bidirectional lexical model in Figure 4.8 that expects \( b_i \) as input to the model.

and feeds it to the decoder part, which is responsible for generating the target sequence. The encoder is composed of one or more stacked recurrent layers. A single-layer encoder is given by:

\[
\begin{align*}
  x_j &= A_s \hat{f}_j \\
  h_j &= \xi(x_j; A_2, h_{j-1}) \\
  c &= h_J,
\end{align*}
\]

where \( A_s \) is the source embedding matrix. \( c \) is the fixed-sized representation of the source sentence, and it corresponds to the recurrent layer output computed using the last word.
4 Neural Networks for Machine Translation

The decoder is composed of one or multiple stacked recurrent layers. A single-layer decoder is given by

\[ y_{i-1} = A_t \hat{e}_{i-1} \]
\[ s_i = \xi(y_{i-1}; A_3, s_{i-1}) , \]

where \( A_t \) is the target word embedding matrix. The source representation vector is used to initialize the decoder state [Sutskever & Vinyals+ 14]

\[ s_0 = g(A_4 c), \]

where \( g \) is a non-linear function applied element-wise, \( A_4 \) is a projection matrix. The target probability distribution is computed using the softmax function

\[ o_i = A_5 s_i \]
\[ p(e_i|e^{i-1}_1, f^j_1) = \text{softmax}(o_i)_{e_i} . \]

Alternatively, [Cho & van Merrienboer+ 14b] propose to make \( o_i \) directly dependent on the previous target embedding \( y_{i-1} \) and the source representation \( c \), in addition to the decoder state \( s_i \). Note that this model is missing the dependence on the alignment variable \( b_i \) included in the previously discussed alignment-based models. One issue with the encoder-decoder architecture is the long time lag between feeding the source word and the prediction of its translation. [Sutskever & Vinyals+ 14] propose to present the source sequence to the network in reversed order, shortening the time lag for the generation of the early part of the target sequence. They found that this trick helps boosting the performance significantly. Another approach would be to feed the source representation vector at each decoder step [Cho & van Merrienboer+ 14b]. This, however, does not mitigate the issue of representing the source sequence by the last recurrent state of that sequence.

4.6.2 Attention-Based Recurrent Neural Network Model

To address the lag problem that the encoder-decoder models have, [Bahdanau & Cho+ 14] proposed to use an attention layer computed dynamically at each target step. The layer computes a probability distribution over the source positions. The intuition behind it is that source positions relevant to the generation of the current translation will have high probability values, while irrelevant words of the source sentence will have (close to) zero values. The output of the distribution is then used to select the source representation to pass to the decoder.

There are many variants of the attention-based recurrent model. The three major common components are the encoder, the attention layer, and the decoder. We describe a model close in architecture to the model presented in [Luong & Pham+ 15] and implemented in Sockeye [Hieber & Domhan+ 17]. The model is shown in Figure 4.10. First, the representation of the source words is computed. The encoder can be unidirectional or bidirectional, and we describe a bidirectional encoder here

\[ x_j = A_x \hat{f}_j \]
\[ h^+_j = \xi(x_j; A_2, h^+_j-1) \]
\[ h^-_j = \xi(x_j; A_3, h^-_j+1) \]
\[ h_j = h^+_j \circ h^-_j , \]
4.6 Attention-Based Neural Network Models

where $\circ$ is the concatenation operator. The source encoding at position $j$ and the last decoder state $s_{i-1}$ are used to compute the attention weights $\alpha_{i,j}$

$$r_{i,j} = v^\top \tanh(A_2 h_j + A_5 s_{i-1} + a)$$  \hfill (4.39)

$$r_i = r_{i,1} \circ r_{i,2} \circ \ldots \circ r_{i,J}$$

$$\alpha_{i,j} = \text{softmax}(r_i)_j$$.
where $\mathbf{a}$ is a one dimensional bias vector, and $\mathbf{v}$ is a one-dimensional projection vector. Note that $\alpha_{i,j}, r_{i,j} \in \mathbb{R}^{1 \times 1}$ are scalars and $r_i \in \mathbb{R}^{J \times 1}$ is a vector. The attention energies described in Equation 4.39 are referred to as additive energies. [Luong & Pham+ 15] describe different multiplicative ways of computing the energies. We mention the generalized dot product approach given by

\[
r_{i,j} = h_j A_4 s_{i-1}. \tag{4.40}
\]

The attention weights are used to obtain the context vector $c_i$

\[
c_i = \sum_{j=1}^J \alpha_{i,j} h_j. \tag{4.41}
\]

In the extreme case of an attention distribution that has all the probability mass assigned to one position $j$, this leads to selecting $h_j$ as the context vector $c_i$ to pass to the decoder. Note the dependence of the context vector $c_i$ on $i$, which is different from the encoder-decoder architecture which computes a single representation $c$ used for all target steps. A representation based on the last target word is given by a recurrent layer

\[
y_{i-1} = A_1 \hat{e}_{i-1}
\]

\[
z_{i-1} = y_{i-1} \circ o_{i-1}
\]

\[
t_{i-1} = \xi(z_{i-1}; A_6, t_{i-2}),
\]

where $o_{i-1}$ is the raw network output right before the output softmax layer from the previous target step. $o_i$ is given by

\[
s_{i-1} = y_{i-1} \circ t_{i-1}
\]

\[
d_i = \xi(c_i \circ t_{i-1}; A_7, d_{i-1})
\]

\[
o_i = g(A_8 (d_i \circ c_i) + b),
\]

where $g$ is a non-linear function, $b$ is a one-dimensional bias vector, and $o_i \in \mathbb{R}^{|V_e| \times 1}$. $o_i$ is computed using a feedforward layer receiving concatenated input. The target posterior distribution is given by

\[
p(e_i|e_{i-1}^j, f_1^J) = \text{softmax}(o_i)|_{e_i}.
\]

The model can be conditioned on the alignment information in many ways. One possibility is to select the context vector $c_i$ using the alignment information

\[
c_i = h_{b_i},
\]

which is equivalent to replacing the attention weights by

\[
\alpha_{i,j} = \begin{cases} 1, & \text{if } j = b_i, \\ 0, & \text{otherwise}. \end{cases}
\]

We will discuss an alternative approach of including alignment in Section 4.7.1. The RNN attention model has sequential dependencies within its layers due to their recurrent nature. The dependencies are illustrated in Figure 4.11. This means that per-layer steps are computed one after the other without parallelization. Next, we will discuss the transformer model architecture, which is based on feedforward layers with self-attention replacing recurrent layers, allowing for heavy parallelization.
4.6 Attention-Based Neural Network Models

Figure 4.11: The dependencies within the encoder and decoder recurrent layers. The figure represents a stacked model with multiple encoder and decoder layers. Within-layer parallelization is not possible due to the sequential processing of recurrent layers. The bidirectional encoder is denoted by the bidirectional arrows.

4.6.3 Multi-Head Self-Attentive Neural Models

Recurrent components pose a problem for parallel computation. This is because they form sequential dependencies. For instance, computing $h_j^+$ in Figure 4.10 requires computing $h_{j-1}^+$ first, which requires $h_{j-2}^+$, and so on. One solution applied by [Vaswani & Shazeer 17] is to replace recurrent layers by self-attentive layers. Self-attention was originally referred to as intra-attention in [Cheng & Dong 16]. Self-attentive layers are computed using an attention component that computes attention weights between two elements of the same sequence. The attention energies presented in Equation 4.39 are computed using a feedforward layer. Instead, self-attentive models are based on energies computed using dot product. The motivation for this is that batched dot-products are more efficient to compute on GPUs. Before discussing self-attention, we introduce the source-to-target attention energies using generalized dot product. The attention energy in Equation 4.39 is reformulated as

\[
\begin{align*}
    r_{i,j}^{(l)} &= (Q^{(l)} s_{i-1}^{(l)})^\top K^{(l)} h_j \\
    r_{i}^{(l)} &= (Q^{(l)} s_{i-1}^{(l)})^\top K^{(l)} [h_1; \ldots; h_J],
\end{align*}
\]

where $Q^{(l)}$ and $K^{(l)}$ are weight matrices in the $l$-th decoder layer. Due to the weight matrices, the two vectors $h_j$ and $s_{i-1}^{(l)}$ can be of different dimensions. The transformer architecture has de-
The transformer model includes a source-to-target attention component per decoder layer. This component is composed of multiple attention layers. These attention components serve to focus the attention of the decoder on the relevant parts of the source sentence. For a network that has \( N \) attention layers per decoder layer and \( L \) decoder layers, a total of \( N \times L \) source-to-target attention layers are used. The concatenation of the representations computed using all source-to-target attention layers in the decoder layer is given by

\[
C_i^{(l)} = c_{i,1}^{(l)} \circ \ldots \circ c_{i,N}^{(l)},
\]

(4.42)

where \( c_{i,n}^{(l)} \) is the aggregated source context vector weighted by the \( n \)-th source-to-target attention layer weights \( \alpha_{i,j,n}^{(l)} \) in the \( l \)-th decoder layer

\[
c_{i,n}^{(l)} = \sum_{j=1}^{J} \alpha_{i,j,n}^{(l)} V_n^{(l)} H_j^{(L)},
\]

where \( V_n^{(l)} \) is a matrix of parameter weights. The output \( C_i^{(l)} \) is passed along with the self-attentive decoder representation \( \tilde{C}_i^{(l)} \) to a subnetwork consisting of a feedforward layer, layer normalization and skip connections. We abstract away this subnetwork for simplicity, and denote the final output of the \( l \)-th decoder layer as \( f(C_i^{(l)}, \tilde{C}_i^{(l)}) \). Note that we ignore components like skip connections and layer normalization in our formulation for better readability. A block diagram illustrating the multi-head source-to-target attention component is shown in Figure 4.12.

A self-attentive layer computes attention weights between positions indexing the same sequence. We drop the attention layer indexing \( n \) in the following equations to simplify the notation. Note, however, that each attention layer has its own separate weights. A self-attentive encoder layer can be computed using the following attention energies:

\[
\tilde{r}_{i,j,\prime}^{(l)} = (Q_{i,j,\prime}^{(l)} h_{j,\prime}^{(l)})^\top K_{i,j,\prime}^{(l)} h_j^{(l)},
\]

\[
\tilde{r}_j^{(l)} = (Q_i^{(l)} h_j^{(l)})^\top K_i^{(l)} [h_1^{(l)}; \ldots; h_J^{(l)}],
\]

where \( j, \prime \in \{1, \ldots, J\} \). \( h_j^{(l)} \) is the word embedding vector for the source word at position \( j \) with positional encoding. For stacked encoders, \( h_j^{(l)} \) is computed using the source representation \( \tilde{H}_j^{(l-1)} \)
4.6 Attention-Based Neural Network Models

![Diagram of the source-to-target multi-head attention component. The linear components indicate multiplication by a weight matrix. The scaled dot-product attention layer computes weighted representations for each of the $N$ attention layers. The representations are concatenated and linearly transformed.](image)

Figure 4.12: The source-to-target multi-head attention component. The linear components indicate multiplication by a weight matrix. The scaled dot-product attention layer computes weighted representations for each of the $N$ attention layers. The representations are concatenated and linearly transformed.

of the previous self-attentive encoder layer at position $j$. $h^{(l)}_j$ is given by

$$h^{(l)}_j = \begin{cases} A_s f_j + p_j, & l = 0, \\ g_{l-1}(H^{(l-1)}_j) & l > 0. \end{cases}$$

where $g_{l-1}$ is a subnetwork that has layer normalization, a feedforward layer, and a residual connection. $p_j \in \mathbb{R}^{D \times 1}$ is the positional embedding vector [Vaswani & Shazeer+ 17]. The $d$-th index of the vector is given by

$$p_{j,2d} = \sin(j/10000^{2d/D}) \quad (4.43)$$
$$p_{j,2d+1} = \cos(j/10000^{2d/D}) \quad (4.44)$$

Such positional encoding is needed to add positional information to the word embedding. This is not needed in recurrent layers because the sequential computation automatically results in positional dependency. The positional encoding can alternatively be learned as network parameters.

The softmax function converts the energies to a probability distribution which is used to weight the sequence representations. The encoder representations are computed as follows:

$$\bar{a}^{(l)}_{j,j'} = \text{softmax}(\tilde{r}^{(l)}_{j,j'})_{j'}$$

$$\tilde{h}^{(l)}_j = \sum_{j'=1}^J \bar{a}^{(l)}_{j,j'} h^{(l)}_{j'}.$$  

The encoder layer output is computed by concatenating the representations of all heads in the layer

$$\tilde{H}^{(l)}_j = \tilde{h}^{(l)}_{j,1} \circ \ldots \circ \tilde{h}^{(l)}_{j,N}. \quad (4.45)$$

Similarly, self-attentive decoder layers can be obtained as follows:

$$\tilde{r}^{(l)}_{i,i'} = (Q_2^{(l)} s^{(l)}_{i-1})^\top K_2^{(l)} s^{(l)}_{i'}$$

$$\tilde{r}^{(l)}_i = (Q_2^{(l)} s^{(l)}_{i-1})^\top K_2^{(l)} [s^{(l)}_1; \ldots; s^{(l)}_{i-1}],$$
for \( i' \in \{1, \ldots, i - 1\} \). Note that self-attention on the decoder side considers positions up to the target position \( i - 1 \) only to preserve causality, as the future target is unknown and yet to be computed. This is different to the encoder case which assumes the full input sequence to be given at all decoding steps. The decoder self-attentive representations \( \tilde{C}_i^{(l)} \) are obtained as follows:

\[
\tilde{\alpha}_{i,i'}^{(l)} = \text{softmax}(\tilde{r}_i^{(l)})|_{i'}
\]
\[
\tilde{c}_i^{(l)} = \sum_{i' = 1}^{i - 1} \tilde{\alpha}_{i,i'}^{(l)} s_i^{(l)}.
\]

The concatenated self-attentive decoder representations are

\[
\tilde{C}_i^{(l)} = f_l'(\tilde{c}_i^{(l)} \circ \ldots \circ \tilde{c}_i^{(l)}),
\] (4.46)

where \( f_l' \) is an abstracted subnetwork.

Analogous to the encoder, the decoder state \( s_i^{(l)} \) used to compute the attention energies at the \( l \)-th decoder layer is given by

\[
s_i^{(l)} = \begin{cases} 
A_l \hat{e}_i + p_i, & l = 0, \\
 f_{l-1}(c_i^{(l-1)}, \tilde{c}_i^{(l-1)}), & l > 0.
\end{cases}
\]
where \( p_i \) is the positional embedding computed using Equations 4.43 and 4.44, and \( C_i^{(l-1)} \) is the concatenation of the source representations weighted using source-to-target attention layers (cf. Equation 4.42). The function \( f^{(l)}_{i-1} \) denotes a subnetwork in the decoder layer.

Self-attentive layers can be used to replace recurrent layers. One advantage is that it is possible to parallelize computation across the sequence. Computing \( \tilde{r}^{(l)}_{j} \) can be done in parallel for all source positions \( j \in \{ 1, \ldots, J \} \) per layer \( l \). The case is the same for computing the decoder attention energies \( \tilde{r}^{(l)}_i \). In contrast, recurrent layers form sequential dependencies which cannot be parallelized. Figure 4.13 illustrates the dependencies within the source and target self-attentive layers.

The transformer model makes extensive use of dropout [Srivastava & Hinton+ 14], layer normalization [Ba & Kiros+ 16], and skip (residual) connections [He & Zhang+ 16]. The model is shown in Figure 4.14.

### 4.7 Alignment-Conditioned Attention-Based Neural Network Models

One issue with the alignment-free models described in the previous section is that there is no guarantee that the attention weights correspond to meaningful word alignment. That is, if the attention weights are used to extract word alignment, the resulting alignment might have words aligned to irrelevant source words, while still producing correct translation. Extracting alignment can be useful in scenarios such as translation override, where the user wants to enforce translation of certain source words. Figure 4.15 (a) shows an example. The transformer baseline incorrectly points to the sentence end symbol “\(</s>\)” when generating the word “strong”. To address this issue, attention-based models that are conditioned on the word alignment are introduced in [Alkhouli & Ney 17] and [Alkhouli & Bretschner+ 18]. These are hybrid lexical models that depend on the attention weights to select the source representation to be passed to the decoder layers, but they also depend on the alignment to make the selection. These models can be considered to lie between the pure alignment-based lexical models presented in Section 4.5 and the alignment-free attention-based models discussed in Section 4.6. Figure 4.15 (b) shows the accumulated attention weights of an alignment-assisted transformer model. Here, the English word “strong” is clearly aligned to the correct Chinese word “强大”. We present both RNN-based and transformer-based alignment-conditioned variants in this section.

#### 4.7.1 Alignment-Biased Attention-Based Recurrent Neural Network Models

In this section we present a variant of the attention-based model presented in Section 4.6.2. This lexical model is proposed in [Alkhouli & Ney 17]. The idea is to compute the attention energies with an extra bias term that depends on an externally provided alignment. In particular, if a source position is aligned to the current target word to be predicted, it is biased using an additional bias vector. Equation 4.39 is modified to include the bias term as follows:

\[
\tilde{r}_{i,j} = v^\top \tanh \left( A_4 h_j + A_5 s_{i-1} + a + \delta_{j,b_i} b \right),
\]

where \( b \) is a learned bias vector and \( \delta_{j,b_i} \) is the Kronecker delta

\[
\delta_{j,b_i} = \begin{cases} 
1, & j = b_i, \\
0, & j \neq b_i.
\end{cases}
\]

This effectively biases the energies corresponding to the aligned source positions, and leaves the other positions without modification. Note that the final attention weights computed using these energies are not directly controlled; therefore, the model still computes dynamic attention weights,
except that there is bias towards the aligned position. Using this approach, there is no guarantee that the attention weights will be high for the biased position. One advantage for biasing attention weights is that it provides an easy way to include external alignment information, while allowing
4.7 Alignment-Conditioned Attention-Based Neural Network Models

Figure 4.15: An example from the BOLT Chinese→English system. The figures illustrate the accumulated attention weights of the baseline transformer model described in Section 4.6.3 (a), and the alignment-assisted transformer model to be described in Section 4.7.2 (b). The maximum accumulated attention weight in the baseline transformer model points incorrectly to the sentence end symbol “<</s>” when generating the translation “strong”. On the other hand, the alignment-conditioned transformer model correctly points to the corresponding Chinese word.

the attention weights not to be necessarily dominated by the alignment. This can be helpful when the alignment is noisy.

Another variant is to use a bias term that includes the aligned source word representation $h_{bi}$

$$r_{i,j} = v^T \tanh \left( A_4 h_j + A_5 s_{i-1} + a + \delta_{i,b_i} A_9 h_{bi} \right),$$

where $A_9$ is a weight matrix. This variant makes the bias term dependent on the source word, while the energy in Equation 4.47 has a constant bias term independent of the source word.

Figure 4.16 illustrates the dependencies in the RNN attention model with alignment bias. The decoder state is computed using biased attention weights. Since the decoder is recurrent, this implies complete dependence on the alignment history. Such dependence is also referred to as a high-order dependence with respect to the alignment variable.

4.7.2 Alignment-Assisted Multi-Head Self-Attentive Neural Network Models

Multi-head self-attentive transformer models described in Section 4.6.3 use dot-product attention layers; therefore, introducing the alignment information as a bias term to the attention energy is not possible in the same way presented in Section 4.7.1. In addition, these models include multiple source-to-target attention layers in each decoder layer. [Alkhouli & Bretschner + 18] propose to add an extra alignment layer in each decoder layer. The alignment layer output is a one-hot distribution with a probability of 1 assigned to the aligned position

$$\alpha^{(l)}_{i,j} = \begin{cases} 1, & j = b_i, \\ 0, & j \neq b_i. \end{cases}$$

Since attention weights are used to weight the source representations, the alignment layer selects the aligned source representation $H^{(L)}_{b_i}$. The source representation is concatenated to the representations computed using the other attention layers in the same decoder layer

$$C^{(l)}_i = c^{(l)}_{i,1} \circ ... \circ c^{(l)}_{i,N} \circ V^{(l)}_{N+1} H^{(L)}_{b_i}.$$
4 Neural Networks for Machine Translation

Figure 4.16: The dependencies within the encoder and decoder recurrent layers including alignment bias. Since the attention weights are used to compute the decoder state, and the decoder is recurrent, the model is dependent on all previous alignment points.

Compared to Equation 4.42, adding the alignment layer corresponds to augmenting the representations with the aligned source representation. The matrix $V_{N+1}^{(l)}$ is used to reduce the dimension of the source representation. Figure 4.17 illustrates the dependencies within the alignment-assisted transformer. Figure 4.18 illustrates the alignment-assisted multi-head source-to-target attention component in a block diagram.

4.7.3 Self-Attentive Alignment Model

[Alkhouli & Bretschner+ 18] propose to leverage self-attentive layers to replace recurrent layers in the bidirectional recurrent alignment model described in Section 4.5.7. The self-attentive alignment model does not use any attention heads. Instead, it has only one alignment head per decoder layer that selects the encoder state in the last encoder layer $H_{b_{i-1}}^{(L)} = g_e(H_{b_{i-1}}^{(L)})$ according to the previous alignment point $b_{i-1}$. This leads to the same source representation $C_{i}^{(l)} = H_{b_{i-1}}^{(L)}$ for all decoder layers $l \in \{0, ..., L\}$, where no attention heads are concatenated. The self-attentive encoder and decoder representations are computed as in the lexical model, and the output layer is changed to predict the source jumps $\Delta_i = b_i - b_{i-1}$.

4.8 Training

Training neural networks refers to estimating the neural network parameters. This typically involves a high-dimensional non-convex optimization problem that has no closed-form analytical solution. The problem is even further complicated by the fact that hundreds of millions of parameters are to be estimated using relatively limited amount of training data. It is in fact not
Figure 4.17: The dependencies within the encoder and decoder self-attentive layers in an alignment-assisted transformer model. The multi-head attention computation at target step \( i \) is directly dependent on the alignment information \( b_i \). Stacking two or more self-attention layers in the decoder makes step \( i \) indirectly dependent on the alignment path \( b_i^{-1} \). This together with the direct dependence on \( b_i \) make the prediction of word \( e_i \) dependent on \( b_i \). In comparison to recurrent layers where one layer is sufficient to induce a dependence on the full history, the depicted self-attentive structure requires at least two layers to have a dependence on the full alignment history. This is because, unlike recurrent layers, self-attentive layers have no dependence on previous steps of the layer \( l \) itself. Instead, there is a dependence on the previous decoder steps of the previous layer \( l - 1 \).

The most common training criterion for neural networks is the cross entropy function. For a given corpus of \( S \) parallel sentences

\[
\left[ ((e_s)^{I_s}_1, (f_s)^{J_s}_1) \right]_{s=1}^S,
\]

where \((e_s)^{I_s}_1\) and \((f_s)^{J_s}_1\) refer respectively to the target and source sentences of the \( s \)-th sentence pair, the cross-entropy criterion is given by

\[
F_\theta = \sum_{s=1}^S \log p((e_s)^{I_s}_1| (f_s)^{J_s}_1; \theta)
= \sum_{s=1}^S \sum_{i=1}^{I_s} \log p((e_s)_i | (e_s)^{i-1}_1, (f_s)^{J_s}_1; \theta),
\]
where $\theta$ denote the free parameters to be estimated. Note that the chain rule of probability is applied to get the word-level probability in the second step. This equation includes all training samples in the objective function, which is referred to as batch training. In practice, the training data is split into mini-batches of size $B$. A mini-batch is typically much smaller than the training set, i.e. $B \ll S$, and the objective function is repeatedly optimized on the mini-batches, instead of the whole training set at once. Batch training is restored in the extreme case of $B = S$. The most common optimization algorithms are variants of stochastic gradient descent (SGD), which we describe next.

### 4.8.1 Stochastic Gradient Descent

Stochastic gradient descent [Robbins & Monro 51] is an iterative stochastic optimization algorithm widely applied to train neural networks. It involves computing the gradients of a pre-defined objective function like the cross-entropy function with respect to the parameters. The algorithm computes new parameter estimates by shifting the old parameter values in a direction opposite to the gradient direction in the parameter space. Algorithm 1 describes the general outline behind stochastic gradient descent. The algorithm is stochastic because it selects the training mini-batches randomly, which takes place when shuffling the training data. This is in contrast to standard (batch) gradient descent, which computes the gradient using the full training set, making it non-stochastic.

Updating the network parameters requires the gradients of the training objective function with respect to the parameters. The gradients are computed using the backpropagation algorithm [Rumelhart & Hinton 86], which is based on the repetitive application of the chain rule of calculus. Recurrent neural networks are trained using a variant called backpropagation through time [Werbos 88, Werbos 90, Williams & Peng 90, Williams & Zipser 95], which unfolds the recurrent neural network over time then applies backpropagation on the unfolded network.

### 4.8.2 Adaptive Moment Estimation (Adam)

There are several expansions and variants of optimizer algorithms based on SGD, including adaptive moment estimation (Adam) [Kingma & Ba 15], Adagrad [Duchi & Hazan 11], Adadelta...
Algorithm 1: Pseudocode for stochastic gradient descent.

Input: training data train, development data dev, parameters $\theta$, learning rate
Output: updated parameters $\theta$

1: repeat
2: shuffle train and split into mini-batches
3: for all mini-batches do
4: compute $\nabla F_\theta$ for the mini-batch
5: update parameters: $\theta \leftarrow \theta - \eta \nabla F_\theta$
6: update learning rate $\eta$
7: end for
8: compute convergence criterion on dev
9: until convergence

[Zeiler 12], and RMSProp [Hinton & Srivastava 12]. Unlike the standard SGD that uses a global learning rate for all parameters, these algorithms compute parameter specific learning rates which are updated continuously during training. Adam is one of the most popular optimization algorithms as of the time of writing this dissertation. In [Bahar & Alkhouli 17], the authors argue that using Adam with some learning rate annealing scheme speeds up convergence to good local optima.

Adam accumulates a decaying average of previous squared gradients. At training step $t$, given the parameters $\theta$, and the exponential decay rate for the first moment $\beta_1$ as a hyperparameter, the first moment decaying average is computed as

$$
g = \nabla F_\theta \quad m_t = \beta_1 m_{t-1} + (1 - \beta_1)g \quad \hat{m}_t = \frac{m_t}{1 - \beta_1}.
$$

Adam also keeps track of the decaying mean of squared gradients $n_t$. Given the exponential decay rate for the second moment $\beta_2$ as a hyperparameter, the second moment decaying average is computed as follows:

$$
n_t = \beta_2 n_{t-1} + (1 - \beta_2)g^2 \quad \hat{n}_t = \frac{n_t}{1 - \beta_2}.
$$

The new parameters are updated as follows:

$$
\theta \leftarrow \theta - \frac{\eta}{\sqrt{\hat{n}_t} + \epsilon} \hat{m}_t.
$$

Note that the effective learning rate $\eta/(\sqrt{\hat{n}_t} + \epsilon)$ is potentially different for each parameter, since $\hat{n}_t$ is computed using the gradient of the parameter to be updated. This is different from stochastic gradient descent, which applies the same global learning rate for all parameters. The typical values for the hyperparameters are $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

4.9 Contributions

The feedforward alignment model in Section 4.5.2 is proposed in [Alkhouli & Bretschner 16]. The author of this dissertation contributed the idea and the implementation of this model. The
model is implemented into Jane. The model architecture is based on the feedforward lexicon model of [Devlin & Zbib + 14], except for the output layer, which is computed over alignment jumps instead of words. The second difference is in using the previous alignment point to predict the jump to the next alignment point, while the lexicon model uses the current alignment point to compute the word probability. The recurrent neural network lexicon model (Section 4.5.5) and the bidirectional recurrent lexicon model that pairs aligned source and target word embeddings (Section 4.5.6) are proposed in [Sundermeyer & Alkhouli + 14]. Martin Sundermeyer and Joern Wuebker contributed the unidirectional lexical model. Martin Sundermeyer proposed using bidirectional layers to capture full source context. The author of this dissertation designed and implemented the bidirectional model into rwthlm. The author also implemented the n-best rescoring functionality into the toolkit. In [Alkhouli & Rietig + 15], the author proposes a variant of the bidirectional model that maintains the original order of the source sentence, but does not include recurrency on the target sequence. The model can be applied during decoding. The author came up with the idea, designed, and implemented the model and integrated it into the Jane decoder. In this dissertation, the author proposes another more expressive variant that also includes recurrency over the target sequence. The author contributed the idea, model design and implementation into the Sockeye toolkit. The model was implemented into Jane by Mohammad Hethnawi during his Master thesis, which was supervised by the author of this dissertation. All these variants are described in Section 4.5.6. Section 4.5.7 describes the bidirectional alignment model proposed in [Alkhouli & Ney 17]. The author contributed the idea, model design and implementation into Sockeye. Mohammed Hethnawi implemented it into Jane. The paper also describes the alignment-biased attention-based recurrent model (Section 4.7.1). Biasing the attention component using the alignment information is completely contributed by the author. The multi-head self-attentive lexicon and alignment models are proposed in [Alkhouli & Bretschner + 18]. The models are described in Sections 4.7.2 and 4.7.3. The author proposed the idea and model design. Gabriel Bretschner implemented the models into Sockeye during his Master thesis, which was supervised by the author.
This chapter discusses neural machine translation, which is the state-of-the-art paradigm as of the time of writing this dissertation. The transition to neural machine translation started in 2014, only to become widely adopted in later years in the industry and in machine translation evaluation campaigns like the Conference on Machine Translation (WMT)\textsuperscript{1} and the International Workshop on Spoken Language Translation (IWSLT). Chapter 4 already introduced the neural network model architectures used in neural machine translation. In this chapter, we will discuss the search procedure for standard neural machine translation and for alignment-based neural machine translation. We will discuss the benefits of the alignment-based approach and present results comparing different neural model architectures.

\section{Introduction}

Neural machine translation refers to machine translation that relies on one or more neural network models to generate translation \citep{Kalchbrenner:13, Bahdanau:14, Sutskever:14, Cho:14a}. This is in contrast to phrase-based systems that can use neural network models to score translation hypotheses generated by other underlying models. We characterize the following differences between the two approaches:

- **Neural machine translation** generates translation hypotheses depending on neural network scores. Phrase-based machine translation generates hypotheses using count-based phrase models. Neural networks are used within phrase-based systems to score translation hypotheses that are already generated, whether scoring happens directly in decoding or offline post decoding in $N$-best rescoring.

- **Neural network training** used in neural machine translation is performed end-to-end. This means that no intermediate training steps are required to generate information needed to train the neural networks. Only the source and target sentence pairs are used to train the models. Phrase-based systems, on the other hand, are based on phrase models which are trained depending on the word alignment. Since the word alignment is usually not readily available, a separate step of word alignment training is run on the parallel training corpus first. The word alignment is then used to construct the phrase models.

- **Another difference** between the two approaches is consistency between the training phase and the decoding phase. Standard neural machine translation scores the translation hypotheses it generates during decoding in the same way the models are used to score the training sentence pairs in the training phase. Phrase-based systems often include a combination of models that are trained independently. The translation score is a combination of these

\footnotesize\textsuperscript{1}Formerly known as the Workshop on Statistical Machine Translation.
model scores. This creates discrepancy between training and decoding in the sense that decoding does not directly correspond to training.

We note that standard neural machine translation does have some differences between training and decoding, such as length normalization and using an evaluation metric (e.g. BLEU) that is different from the training objective function (e.g. cross-entropy). We will discuss these details throughout this chapter. Another difference is that training is performed using the reference target words, while decoding is performed using the hypothesized target words. Nevertheless, neural machine translation is much more consistent and its training is conceptually much simpler than the phrase-based paradigm. This simplicity is mainly due to a shift in complexity from decoding to modeling: the neural model architecture is significantly more complex than the count-based models used in phrase-based systems. There is still a lack for a mathematical framework to rely on when designing these networks; therefore, neural architectures require careful design and experimentation. Network design is mostly dependent on trial and error with extensive experimentation to find the most suitable architecture design, in terms of capturing the complex linguistic phenomena in natural language. While a simple neural network of one hidden layer is a universal function approximator that, in theory, should be able to model any functional dependency [Hornik 91], in practice, the quality of these models is based on whether a good estimation of the network weights can be found easily, and whether they actually do capture the desired linguistic dependencies; therefore, designing neural networks usually implies finding architectures that are feasible to train in practice.

We also note that there are hybrid approaches that integrate phrase-based models into neural machine translation search [Dahlmann & Matusov + 17]. These systems require significant changes to the decoding algorithm, and due to the large number of models included, they also require careful model weight tuning, making the setup of these systems more complex. Our focus in this chapter is on standalone neural machine translation systems that do not have extra non-neural components. In Chapter 6, we will discuss applying neural networks to phrase-based systems.

5.2 State of The Art

In this chapter, we describe alignment-based systems that utilize alignment during training and decoding [Alkhouli & Bretschner + 16] (Section 5.6). The alignment-based systems are inspired by the HMM framework, but instead of having simple dependencies, neural network models are used to capture long, even unbounded, source and target context. [Cohn & Hoang + 16] incorporate structural biases based on traditional word-based translation models into the attention component directly. The biases include absolute positional bias, where the attention values are computed using the source and target word positions. Inspired by the HMM concept, they include Markov conditioning by making the attention values directly dependent on the attention weights of the previous target word. They also include the concepts of word fertility based on attention weights, and encourage attention symmetry between the source-to-target and target-to-source attention directions to achieve bilingual symmetry. [Arthur & Neubig + 16] include lexical probabilities to bias attention. [Chen & Matusov + 16, Mi & Wang + 16] add to the training objective function an extra term dependent on the alignment to guide neural training. This is only applied during training but not during decoding. The beam search algorithm used in neural machine translation is described in Section 5.5 [Bahdanau & Cho + 14, Sutskever & Vinyals + 14, Cho & van Merrienboer + 14a]. [Alkhouli & Bretschner + 16] modify the beam search algorithm for alignment-based models. The algorithm is presented in Section 5.6.3.

Deriving neural models for translation based on the HMM framework can also be found in [Yu & Blunsom + 17, Yu & Buys + 16]. The work uses a monotonous alignment model, and training is done by marginalizing over the alignment hidden variables, which is computationally
expensive. In this thesis, we use non-monotonous alignment models. In addition, we explore using pre-computed Viterbi alignments which speeds up neural training. In [Yu & Blunsom + 17], alignment-based neural models are used to model alignment and translation from the target to the source side (inverse direction), and a language model is included in addition. They present results on a small translation task. We present results on translation tasks containing tens of millions of words. We do not include a language model in any of our systems.

There have been several attempts to include syntax in neural machine translation. [Bastings & Titov + 17] use a dependency parse tree of the source sentence to construct a graph convolutional encoder. [Aharoni & Goldberg 17] translate into linearized, lexicalized constituency trees. These trees are obtained using independent parsing toolkits.

There are many ways to consider when training alignment-based systems. [Wang & Alkhouli + 17, Wang & Zhu + 18] use Expectation-Maximization training. Instead of using Viterbi alignment, the sum over all possible alignment paths is computed. This utilizes the forward-backward algorithm, which is feasible only when the dependency on the alignment path is first-order. [Wu & Shapiro + 18] also introduce the alignment as a latent variable, making the alignment variable only dependent on the source sentence and target history words. To allow for efficient marginalization sum over the alignment variable, they keep the current alignment value independent of previous alignment values. Our models include long-context dependency on the alignment path; therefore, the forward-backward algorithm cannot be applied efficiently. We resort to using pre-computed Viterbi alignments. We also study re-aligning the training data using forced-alignment training (Section 5.6.2). Forced-alignment training is performed in [Wuebker & Mauser + 10, Peitz & Mauser + 12] to re-align the training data during building phrase-based systems. In this thesis, we describe a conceptually similar approach to re-align data used to train neural systems.

There are various approaches to perform constrained translation. One possibility is including this information in training, but this requires knowing the constraints at training time [Crego & Kim + 16]. Post-processing the hypotheses is another possibility, but this comes with the downside that offline modification of the hypotheses happens out of context, which can lead to inconsistencies due to linguistic phenomena such as morphological inflection. A third possibility is to do constrained decoding [Hokamp & Liu 17, Chatterjee & Negri + 17, Hasler & De Gisper + 18, Post & Vilar 18]. This does not require knowledge of the constraints at training time, and it also allows dynamic changes of the rest of the hypothesis when the constraints are activated. [Dinu & Mathur + 19] train the neural model to inject terminology terms in the target without modifying the decoding algorithm. They inject the terminology in the sentence, and augment the source sentence with another stream indicating code-switching between the original source words and the target terminology words. We perform experiments where the translation is guided online during decoding (Section 5.7.12). We focus on the case where translation suggestions are to be used when a word in the source sentence matches the source side of a predefined dictionary entry. We use the alignment-based framework to detect when the source word of interest is being translated.

5.3 Model

Neural machine translation uses neural network model structures such as the encoder-decoder architecture (Section 4.6.1), the attention-based recurrent neural network (Section 4.6.2), and the multi-head self-attentive neural network (Section 4.6.3). All these architectures have a sub-network called the encoder that computes a representation of the source sentence, and a decoder sub-network that generates the target sequence. The attention-based models introduce one or more attention components that compute a probability distribution over the source positions, determining which encoder source representations should be focused on at a given decoding step.
Neural Machine Translation

[Bahdanau & Cho+ 14] report that the use of the attention components outperforms pure encoder-decoder architectures. This is due to the fact that encoder-decoder models encode the whole source sentence in a fixed-size representation that is used to generate the target sentence. On the other hand, the use of the attention mechanism allows maintaining source position-dependent representations and allows accessing any of them at any decoding time step. Intuitively, it is easier to generate a translation while focusing on certain source words than to generate it from a sentence-level representation, specially for long source sentences; therefore, we only focus on attention-based models in this chapter.

Computing the softmax function for a large vocabulary is a computational bottleneck. There are several solutions to work around this. The class-factored output layer (Section 4.4.7) can be used to limit the computation to a subset of the vocabulary. [Alkhouli & Bretschner+ 16] presented a search algorithm that exploits class factorization to speed up decoding. This solution maintains the original target vocabulary size. Another alternative is to select a short list of the most frequent words, and map all other words to a special unknown token [Bahdanau & Cho+ 14, Sutskever & Vinyals+ 14]. Typical short lists can range in size from 30k to 80k words. This solution reduces the computational effort, but it also discards words from the vocabulary. As a result, rare words do not get correctly translated. One solution for this is to back off to a look-up dictionary when an unknown word is encountered [Jean & Cho+ 15], or to simply copy the source word with the maximum attention weight to replace the unknown token. Another alternative widely adopted in machine translation is the use of subword units, which we describe next.

5.4 Subword Units

[Sennrich & Haddow+ 16] proposed splitting words into subwords. This effectively reduces the vocabulary size since a lot of subwords are shared between words. This also allows reconstructing words from subwords, making the construction of the original vocabulary words also possible. This approach is also appealing because it can model out-of-vocabulary words not seen in training data by concatenating subwords. For instance, if the words “listening” and “reader” are in the training data, and if “listening” is split into two subwords “listen@@ ing”, and “reader” is split into “read@@ er”, then constructing the word “reading” from the two subwords “read@@ ” and “ing” is possible, even if “reading” does not occur in the training data.

[Sennrich & Haddow+ 16] proposed to use a data compression technique, called Byte Pair Encoding (BPE) [Gage 94], to split the vocabulary. Starting from characters as initial units, the construction algorithm merges the most frequent adjacent unit pairs in the training data iteratively, creating new subword units. The number of merge operations is usually limited between 20k and 50k. As a result, word splitting targets rare words more than frequent words, resulting in rare words being split, and frequent words having fewer or no splits. [Sennrich & Haddow+ 16] compared using BPEs as subword units to using a back-off dictionary to translate unknown words, and showed that BPE subword units perform better. In this work, we apply BPE splitting in some of our experiments to the source and target corpora. We will highlight when BPE is used.

5.5 Search

The problem of search in neural machine translation is relatively simpler than that of phrase-based systems, mainly because basic neural machine translation systems rely on a single model.
to generate translation. During search, a solution to the following equation is sought:

\[
J^{1} \rightarrow \hat{e}_{I}^{1}(f_{J}^{1}) = \operatorname{argmax}_{I,e} \left\{ \log p(e_{I}^{1} | f_{J}^{1}) \right\}
\]

where the probability \( p(e_{I}^{1} | f_{J}^{1}) \) is decomposed using the chain rule of probability, and \( p(e_{i}^{1} | e_{i-1}^{1}, f_{J}^{1}) \) is the distribution computed by the end-to-end neural network model. The goal of search is to find the translation sentence that has the maximum probability according to the model. Equation 5.2 will favor shorter sentences since longer sentences are less probable than shorter ones. To remedy this, length normalization is used [Cho & van Merrienboer\textsuperscript{+} 14a, Wu & Schuster\textsuperscript{+} 16]. The modified decision rule using length normalization which is used in this work is given by

\[
J^{1} \rightarrow \hat{e}_{I}^{1}(f_{J}^{1}) = \operatorname{argmax}_{I,e} \left\{ \frac{1}{I} \sum_{i=1}^{I} \log p(e_{i}^{1} | e_{i-1}^{1}, f_{J}^{1}) \right\}
\]

Search has a combinatorial complexity as it enumerates all possible target sentences. In practice, beam search is used as an approximation to limit the number of candidate hypotheses to a constant number at each time step. Algorithm 2 describes how beam search is used in neural machine translation. InitialBeam initializes the partial hypotheses with the sentence begin symbol \(<s>\), and the initial scores are set to zero. Since the target length is unknown, translation is limited to a maximum of \( \delta \cdot J \) steps, where \( \delta \) is a factor usually set to a value between 1.5 and 2. Another possibility is to terminate the for loop over the target positions when all hypotheses have the sentence end symbol \(<</s>\) (lines 17–19). This is different to phrase-based decoding, where the termination criterion is to have all source positions covered. In neural machine translation, there is no well-defined notion of coverage, due to the lack of an explicit alignment concept. All finished hypotheses are collected in \( F \) (line 11) and removed from the beam to allow for a maximum beam capacity (line 12). The final score is computed by normalizing over the target length (line 10) so that long hypotheses are not disadvantaged.

**5.6 Alignment-Based Neural Machine Translation**

Standard neural machine translation does not model alignment explicitly. Instead, in attention-based models, one or more attention layers are used to compute a soft distribution that can be interpreted as alignment. However, there is no guarantee that attention weights will have high values for the source word being translated at any decoding step. There is nothing explicit in the training phase that constrains the network to produce focused attention weights; therefore, relying on attention weights is not guaranteed to yield meaningful word alignment. Alignment-based neural machine translation is a variant of neural machine translation that aims to address
5 Neural Machine Translation

```
1: function Translate(fJ1)
2:  hyps ← InitialBeam(fJ1)                  ▷ Initial beam
3:  scores ← {0, ..., 0}                    ▷ Initial scores
4:  F ← ∅                                ▷ Set of finished hypotheses
5:  for time step i = 1, ..., δ · J do
6:     (hyps, scores) ← Advance(hyps, scores)
7:     done ← True
8:     for (hyp, score) ∈ (hyps, scores) do
9:         if hyp has <s> then
10:             finalScore ← score
11:             F ← F ∪ (hyp, finalScore)
12:         else
13:             done ← False
14:         end if
15:     end for
16:     if done then
17:         break
18:     end if
19:  end for
20:  return F
21: end function

22: function Advance(hyps, scores)
23:  (hyps, localScores) ← LexicalDistribution(hyps) ▷ batched for all hypotheses
24:  scores ← scores + localScores
25:  (hyps, scores) ← GetBest(hyps, scores, beamSize)
26:  return (hyps, scores)
27: end function
```

Algorithm 2: Beam search for neural machine translation.

this problem. The concept was first introduced in [Alkhouli & Bretschner+ 16], where the idea is to decompose translation into a generative story of two steps

1. Align the source word to be translated.
2. Generate the lexical translation conditioned on the aligned source word, along with other context.

Formally, the posterior distribution \( p(e^*_1|f^*_1) \) is decomposed to an alignment model and a lexical model

\[
p(e^*_1|f^*_1) = \sum_{b^*_1} p(e^*_1, b^*_1|f^*_1) = \sum_{b^*_1} \prod_{i=1}^{I} \left( p(e_i|b_i, e_i^{i-1}, e_i^{i-1}, f^*_1) \cdot p(b_i|b_i^{i-1}, e_i^{i-1}, f^*_1) \right),
\]

where the alignment sequence \( b^*_1 = b_1...b_i...b_I \) is introduced as a hidden variable, and a marginalization sum over all possible alignment sequences is used. The joint distribution is decomposed using the chain rule of probability into a lexical model and an alignment model. The alignment

\[
52
\]
model computes the probability of $b_i$, which is the source position to be translated at target step $i$. The lexical model is conditioned on $b_i$, and it computes the probability of the translation word $e_i$. Both the lexical and alignment models are conditioned on rich source and target context. The full source sentence $f_1^J$ is used as context, in addition to the full target history $e_1^{i-1}$. The difference between the alignment model and the lexical model is in the conditioning on the current alignment point $b_i$. Only the lexical model is conditioned on the current alignment. The alignment model predicts this alignment point and therefore cannot be conditioned on it. Note the similarity to Equation 3.7, with the main difference being the direction of translation. Here, we model translation from the source side to the target side, i.e. the probability of the target sequence is computed conditioned on the source sequence. In Equation 3.7, translation follows the inverse direction from the target sequence to the source sequence. The Viterbi approximation [Viterbi 67] can be used to approximate the marginalization sum in Equation 5.5

\[
p(e_1^I|f_1^J) \approx \max_{b_1^I} \prod_{i=1}^I p(e_i|b_i, b_i^{i-1}, e_1^{i-1}, f_1^J) \cdot p(b_i|b_i^{i-1}, e_1^{i-1}, f_1^J). \tag{5.6}
\]

Next, we give an overview of the lexical and alignment models.

### 5.6.1 Models

We described different alignment-based lexical models in Chapter 4. Alignment-based lexical models are similar to attention-based lexical models in that they include encoder and decoder components, and possibly one or more attention layers. The main difference is that the model requires external alignment information as input. In some variants, the alignment is used to select the encoder representation directly (cf. Section 4.5.6) and pass it to the decoder component. In alignment-biased RNN attention models (cf. Section 4.7.1), the attention weights are biased using the alignment signal. Still, computing attention weights is done dynamically. This has the benefit of using both the certainty of the alignment information and the flexibility of attention weights. If the model is uncertain on its own, the external signal helps to bias it. If it is certain, the external signal can be overridden. Lexical models with multi-head attention components augment the soft-weighted encoder representations with the hard selection of the alignment-indexed encoder representation. Again, the attention weights can override the alignment signal when they are large enough. An architecture comparison between the different lexical models we experiment with in this dissertation is shown in Table 5.1.

Alignment-based neural machine translation introduces a dedicated neural alignment model that computes a probability distribution over the source positions. This alignment model is applied in a different way compared to the attention layer. While the attention weights are used to weight the encoder representations, the alignment model score is used directly to score translations, as shown in Equation 5.6. We use three different alignment model architectures in this work, as described in Table 5.2.

A comparison between the feedforward, bidirectional RNN, and transformer models in terms of context is presented in Table 5.3. We note that both the bidirectional RNN and transformer models have access to the same rich dependencies, while the feedforward model has access to limited context. The same applies to the corresponding alignment model variants as shown in Table 5.4. Note that the alignment models are conditioned not on the current alignment point which they have to predict, but on the previous alignment point.

### 5.6.2 Training

We describe three different training recipes to train the alignment-based lexical and alignment models.
Table 5.1: A comparison between the different lexical models we experiment with. The comparison is between the feedforward neural network (FFNN) model (cf. Section 4.5.1), the bidirectional RNN alignment-based model (cf. Section 4.5.6), the RNN attention model (cf. Section 4.6.2), the alignment-biased RNN attention (cf. Section 4.7.1), the multi-head self-attentive transformer model (cf. Section 4.6.3), and the alignment-assisted transformer (cf. Section 4.7.2).

<table>
<thead>
<tr>
<th>lexical model</th>
<th>encoder</th>
<th>decoder</th>
<th>attention</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFNN alignment</td>
<td>feedforward</td>
<td>feedforward</td>
<td>-</td>
</tr>
<tr>
<td>RNN alignment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNN attention + alignment</td>
<td>bidirectional recurrent</td>
<td>recurrent</td>
<td>single-head</td>
</tr>
<tr>
<td>transformer + alignment</td>
<td>self-attentive</td>
<td>self-attentive</td>
<td>multi-head</td>
</tr>
</tbody>
</table>

Table 5.2: A comparison between the different alignment models we experiment with. The comparison is between the feedforward alignment model (cf. Section 4.5.2), the bidirectional RNN alignment model (cf. Section 4.5.7), and the self-attentive alignment model (cf. Section 4.7.3). Note that no attention component is used in any of these models.

<table>
<thead>
<tr>
<th>alignment model</th>
<th>encoder</th>
<th>decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFNN</td>
<td>feedforward</td>
<td>feedforward</td>
</tr>
<tr>
<td>RNN</td>
<td>bidirectional recurrent</td>
<td>recurrent</td>
</tr>
<tr>
<td>self-attentive</td>
<td>self-attentive</td>
<td>self-attentive</td>
</tr>
</tbody>
</table>

Fixed-Alignment Training

Given a word-aligned training corpus. The lexical and alignment models can be trained independent from each other. Since the training corpora are usually not word-aligned, this requires generating word alignment as a first step. Word alignment can be generated using IBM1/HMM/IBM4 training, which is implemented in tools like GIZA++. Fastalign [Dyer & Chahuneau+ 13] is another tool that is based on a reparameterization of the IBM2 model, hence, it is faster compared to the more complex IBM4 training. However, the IBM2 lexicon is only conditioned on the source word; therefore, the word alignment is usually of worse quality compared to IBM4 word alignment. In this work, we rely on GIZA++ to generate the word alignment. We use symmetrization by computing source-to-target and target-to-source word alignment, and then merging them using the grow-diagonal-final-and heuristic [Koehn & Och+ 03]. This alignment can contain many-to-one and one-to-many word alignment cases. We convert the alignment such that each target word is aligned to exactly one source word as proposed in [Alkhouli & Bretschner+ 16]. We follow the following heuristics [Devlin & Zbib+ 14]:

- Unaligned target words inherit the alignment of the closest target word, preferring left to right.
- Multiple-aligned target words select the middle source word, and ignore all other alignment points.
Table 5.3: A comparison between the feedforward lexical model (Section 4.5.1), the bidirectional RNN lexical model (Sections 4.5.6, 4.7.1) and the transformer lexical model (Section 4.7.2) in terms of the source, target and alignment context. All models are conditioned on some alignment information. The feedforward lexical model uses a limited context of $2m + 1$ source words, while the RNN and transformer models use the full source sentence. The feedforward lexical model is also limited to $n$ target history words, compared to the full target history used by both the RNN and the transformer models. Finally, the feedforward model is only conditioned on the current alignment point, while the other models are conditioned on the full previous alignment path. The RNN and transformer models have access to richer dependencies than the feedforward model. During search, the bidirectional RNN and transformer models maintain layer states that are needed for efficient computation of the next step. The feedforward model, on the other hand, is computed from scratch each time, and has smaller memory footprint during decoding.

<table>
<thead>
<tr>
<th>alignment model</th>
<th>notation</th>
<th>source</th>
<th>target</th>
<th>alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFNN alignment</td>
<td>$p(e_i</td>
<td>e_{i-1}^{i-1}, f_{b_i}^{i+m})$</td>
<td>$f_{b_i-m}...f_{b_i+m}$</td>
<td>$e_{i-n}...e_{i-1}$</td>
</tr>
<tr>
<td>RNN alignment</td>
<td>$p(e_i</td>
<td>e_{i-1}^{i-1}, f_j, b_i)$</td>
<td>$f_1...f_J$</td>
<td>$e_{1...e_i-1}$</td>
</tr>
<tr>
<td></td>
<td>RNN attention + alignment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>transformer + alignment</td>
<td>$p(e_i</td>
<td>e_{i-1}^{i-1}, f_j, b_i)$</td>
<td>$f_1...f_J$</td>
<td>$e_{1...e_i-1}$</td>
</tr>
</tbody>
</table>

Table 5.4: A comparison between the feedforward alignment model (Section 4.5.2), the bidirectional RNN alignment model (Section 4.5.7) and the self-attentive alignment model (Section 4.7.3) in terms of the source, target, and alignment context. All models are conditioned on some alignment information. The feedforward alignment model uses a limited context of $2m + 1$ source words, while the RNN and transformer models use the full source sentence. The feedforward lexical model is also limited to $n$ target history words, compared to the full target history used by both the RNN and the self-attentive models. Finally, the feedforward model is only conditioned on the previous alignment point, while the other models are conditioned on the full alignment path.

<table>
<thead>
<tr>
<th>alignment model</th>
<th>notation</th>
<th>source</th>
<th>target</th>
<th>alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFNN</td>
<td>$p(\Delta</td>
<td>e_{i-1}^{i-1}, f_{b_i}^{i+m})$</td>
<td>$f_{b_i-1-m}...f_{b_i-1+m}$</td>
<td>$e_{i-n}...e_{i-1}$</td>
</tr>
<tr>
<td>RNN</td>
<td>$p(\Delta</td>
<td>e_{i-1}^{i-1}, f_j, b_i^{j-1})$</td>
<td>$f_1...f_J$</td>
<td>$e_{1...e_i-1}$</td>
</tr>
<tr>
<td>self-attentive</td>
<td>$p(\Delta</td>
<td>e_{i-1}^{i-1}, f_j, b_i^{j-1})$</td>
<td>$f_1...f_J$</td>
<td>$e_{1...e_i-1}$</td>
</tr>
</tbody>
</table>

The cases of one-to-one alignment, unaligned source words, and multiple-aligned source words are left without change. An example of resolving alignment ambiguities is shown in Figure 5.1. After applying the heuristics, we have the following training data:

$$\left[\left((e_s)_1^{I_s}, (f_s)_1^{J_s}, (b_s)_1^{I_s}\right)\right]^S_{s=1}.$$
pollutant · · · · · · · · · · ■ ■ ●
this · · · · · · · · · · · · · ·
of · · · · · · · · · · · · · · · ·
full · · · · · · · · · · · · · · · ·
offspring · · · · · · · · · · · · ●
first · · · · · · · · · · · · · ·
their · · · · · · · · · · · · · ·
pump · · · · · · · · · · · · · ·
mothers ♦ ♦ · · · · · · · · · ·
these ♦ · · · · · · · · · · · ·
source

Figure 5.1: An example showing the conversion of many source to one target alignment. The source words “Mutter Tiere” are aligned to the target word “mothers”. This is resolved by selecting the middle source word. In the case of an even number of source words as in this example, we select the word to the left of the center, which is “Mutter” in this example. The selected alignment is marked in red. Note that “Muttertiere” is a single compound word in German, but it is split here due to a preprocessing step. We also show the assignment of an alignment point to a target word that is originally unaligned. This is marked in blue. The target word “of” inherits its alignment from the neighboring word “full” to the left of it, which is aligned to the source word “voll”. Hence, we consider the word “of” aligned to “voll” as well.

Let \( \theta = \{ \theta_l, \theta_a \} \) be the set of all lexical model parameters \( \theta_l \) and the alignment model parameters \( \theta_a \). The cross-entropy criterion is given by

\[
F_\theta = \sum_{s=1}^{S} \log p \left( (e_s)_1^j, (b_s)_1^j | (f_s)_1^j; \theta \right)
\]

\[
= \sum_{s=1}^{S} \sum_{i=1}^{I_s} \log p \left( (e_s)_i | (e_{s-1})_1^{i-1}, (b_s)_1^j, (f_s)_1^j; \theta_l \right) + \log p \left( (b_s)_i | (e_{s-1})_1^{i-1}, (b_{s-1})_1^{i-1}, (f_s)_1^j; \theta_a \right)
\]

For the case when \( \theta_l \cap \theta_a = \emptyset \), we can separate the training of the two models. The lexical model and alignment model training objective functions are respectively

\[
F_{\theta_l} = \sum_{s=1}^{S} \sum_{i=1}^{I_s} \log p \left( (e_s)_i | (e_{s-1})_1^{i-1}, (b_s)_1^j, (f_s)_1^j; \theta_l \right)
\]

\[
F_{\theta_a} = \sum_{s=1}^{S} \sum_{i=1}^{I_s} \log p \left( (\Delta_{s})_i | (e_{s-1})_1^{i-1}, (b_{s-1})_1^{i-1}, (f_s)_1^j; \theta_a \right),
\]

where we change the alignment model prediction to be the source jump \( (\Delta s)_i = (b_s)_i - (b_{s-1})_i \) instead of \( (b_{s})_i \). This is done to model relative jumps regardless of the absolute source position.

**Forced-Alignment Training**

One drawback of the previous training approach is that it is not end-to-end, as it depends on word alignment generated using other models. Another drawback is that the quality of the
models will depend on the quality of the word alignment generated, which is fixed. Another way of doing training is to generate the word alignment using the neural models. The alignment is then used to train the models, which are used to re-align the data again, and so on. This is an iterative approach that alternates between aligning the data and training the models using the created alignment [Alkhouli & Bretschner+ 16]. The approach can be summarized as follows:

1. Generate initial word alignment (e.g. diagonal alignment, IBM4 alignment, etc.).
2. Start or continue training the lexical and alignment models using the alignment.
3. Re-align the training data using the trained models.
4. Repeat 2-3 until convergence.

The initialization step can use initial IBM4 alignment. If end-to-end training is desired, diagonal alignment can be used. Another possibility is to use random alignment generated using non-trained lexical and alignment models. Given the alignment, the lexical and alignment models are trained as described in the previous subsection. The models do not need to be trained again from scratch. Instead, the previous set of model parameters can be used as initialization for the current training iteration. The trained models are then used to re-align the training data. The problem of re-alignment is similar to that of search, except that the target side is given; therefore, forced-alignment is only concerned with searching for the best alignment path for a given source and target sentence pair. This is done according to the following equation:

\[
(e_1^f, f_1^j) \rightarrow \hat{b}_1^f(e_1^f, f_1^j) = \arg\max_{b_1^f} \sum_{i=1}^{l} \lambda \log p(e_i | e_{i-1}^f, b_1^f, f_1^j) + (1 - \lambda) \log p(\Delta_i | e_{i-1}^f, b_1^{i-1}, f_1^j),
\]

where we introduce the lexical model weight \( \lambda \in (0, 1) \), and the alignment model weight \((1 - \lambda)\). Searching for the alignment path is done using beam search, which we will describe in Section 5.6.3. Since this approach looks for the best alignment path using the maximum Viterbi approximation, we also refer to it as Viterbi training. Note that both lexical and alignment models are used to compute the alignment; therefore, re-aligning the training data can be viewed not only as a way to correct alignment errors, but also as a way of doing joint training of the lexical and alignment models, since the alignment generated using both will be used to train each one of them. A flow chart describing forced alignment training is illustrated in Figure 5.2.

**Expectation-Maximization Training**

Instead of using the maximum approximation in Equation 5.6 to evaluate the models using the best alignment path, we can keep the sum over all possible alignment paths as described in Equation 5.5. In this case, the models can be trained using the expectation-maximization (EM) algorithm [Dempster & Laird+ 77]. The algorithm is iterative, and it consists of two main steps: the expectation step (E) and the maximization step (M). The expectation step refers to computing posterior alignment path probabilities \( p_i(j | e_1^f, f_1^j, \theta) \) and \( p_i(j', j | e_1^f, f_1^j, \theta) \) using the forward-backward (Baum-Welch) algorithm. Computing these probabilities involves both lexical and alignment models. These posterior probabilities represent soft alignment probabilities. This is in contrast to the maximum approach which is based on hard alignment. The maximum approximation training can be considered a special case of having the entire posterior probability mass assigned to one alignment path. The maximization step refers to the parameter estimation, or in our case, the neural network training phase. The difference between Viterbi training and EM training is that the derivative of the network model gets weighted by the posterior probability of
Figure 5.2: A flow chart describing forced-alignment training. An initial alignment is expected, which is used to train the lexical and alignment models. The two models are used to generate translation. Convergence can be determined in many ways, e.g. training can be considered converged if the translation quality does not improve compared to the translation of the previous iteration. Both lexical and alignment models are used to re-align the training data and the new alignment is used to re-train the models.

the alignment points [Wang & Alkhouli+ 17]. The derivatives of the training objective functions of the lexical and alignment models are respectively given by

$$\frac{\partial F_\theta}{\partial \theta_l} = \sum_{i=1}^{I} \sum_{j=1}^{J} p_i(j|e_i^1, f_1^j; \theta) \cdot \frac{\partial}{\partial \theta_l} \log p(e_i|e_i^{i-1}, b_i = j, f_1^j; \theta_l)$$  \hspace{1cm} (5.8)

$$\frac{\partial F_\theta}{\partial \theta_a} = \sum_{i=1}^{I} \sum_{j'=1}^{J} \sum_{j=1}^{J} p_i(j', j|e_i^1, f_1^j; \theta) \cdot \frac{\partial}{\partial \theta_a} \log p(\Delta_i = j - j'|e_i^{i-1}, b_{i-1} = j', f_1^j; \theta_a).$$  \hspace{1cm} (5.9)

Note that we limit the alignment dependency to the current aligned position $b_i = j$ in the lexical model and to the previous aligned position $b_{i-1} = j'$ in the alignment model case. This is because computing the forward-backward algorithm has an exponential complexity in the order of the alignment; therefore, conditioning the models on the full alignment path $b_i^1$ which is of high order makes using the forward-backward algorithm infeasible. To overcome this, models which have low-order alignment dependencies such as the feedforward lexical and alignment models can be used (cf. Tables 5.3,5.4). In [Wang & Alkhouli+ 17], the authors used feedforward lexical and alignment models. The system in this case has a first-order dependency, since the alignment
5.6 Alignment-Based Neural Machine Translation

The model is only conditioned on the immediate previous alignment point. [Wang & Zhu 18] used the forward-backward algorithm training to train recurrent models. These models also have a first-order dependency on the alignment information, since the alignment information does not go through a recurrent layer. We refer the reader to the discussion in Section 4.5.6 for more details on the model design and its effect on the alignment order. In Viterbi training, there is only one path with a probability of 1, and other paths have a probability of 0; therefore, the sums over source positions \( j \) and \( j' \) in Equations 5.8 and 5.9 collapse to a single term.

We do not investigate EM training in this dissertation. We keep the full-order dependency on the alignment path and use either a fixed alignment to train the models or perform iterative forced-alignment training using the Viterbi maximum approximation.

5.6.3 Search

Search in alignment-based neural machine translation includes both the lexical and the alignment model. It aims to find the maximizing argument for the following formula:

\[
f_{f_1}^{e_1} = \arg\max_{I, e_1} \frac{1}{T} \left( \sum_{i=1}^{T} \lambda \log p(e_i | e_{i-1}^{i-1}, b_i^{i-1}, f_{J_1}^{1}) + (1 - \lambda) \log p(\Delta e_i | e_{i-1}^{i-1}, b_{i-1}^{i-1}, f_{J_1}^{1}) \right)
\]

where \( \lambda \) is the lexical model weight, and \((1 - \lambda)\) is the alignment model weight. The models are combined in a log-linear fashion and the score is normalized by the target length. Note the additional maximization over the alignment path \( b_i^I \) that underlies translation, which does not occur in standard neural machine translation.

The search procedure used in alignment-based neural machine translation was proposed in [Alkhouli & Bretschner 16]. It is an extension of the beam-based search algorithm used for standard neural machine translation. Besides the lexical word hypotheses, search also includes hypothesizing word alignment. Algorithm 3 is a modification of Algorithm 2, which introduces the necessary changes to incorporate alignment into search. The changes are marked in red.

The main change is in line 25, where an extra loop over the source positions is used. This loop hypothesizes alignment positions, and computes lexical model scores that are conditioned on the alignment position \( j \) (line 26). The alignment model is computed once outside the loop in line 24, since it computes a probability of the next source jump (or equivalently next alignment point). The scores of all possible jumps are computed at once in the softmax output layer of the model, hence, the model call is factored out of the for loop. The alignment model computation for each hypothesis is conditioned on the previous alignment point as indicated in the function call, which depends on \texttt{previousAlignments} (plural form used to indicate that this includes the previous alignment point for each of the hypotheses).

Search hypotheses are composed not only of lexical translations but also of alignment hypotheses. Search keeps track of both lexical and alignment hypotheses. The alignment model scores are expected to favor good alignment hypotheses. The beam entry scores are a log-linear combination of lexical and alignment model scores, as shown in line 28. The call to \texttt{GetBest} sorts the combined scores and returns \texttt{beamSize} many hypotheses. Note that both lexical and alignment hypotheses are maintained in a mixed stack. Under these settings, one word translation can occur many times within the beam, as long as it is generated using different alignment positions.

Figure 5.3 shows an excerpt of a search graph. Because there is an alignment underlying the lexical hypotheses, we can easily keep track of a coverage vector, like in phrase-based machine translation. This can be used to enforce certain constraints, like how many times a source position can be translated. Note that, unlike the case of phrase-based machine translation, we do not enforce full coverage to terminate a hypothesis. A hypothesis is considered final if it ends with the sentence end symbol \(<\text{/s}>\). Another difference to phrase-based machine translation is

that translation is generated one target (sub)word at a time, while phrase-based translation can
generate multi-word phrases in a single decoding step (cf. Figure 6.4). Final hypotheses are kept
track of and removed from the beam. The final translation can be generated by backtracking the
terminal node with the best final score.

Algorithm 3 handles the different word alignment cases as follows:

• Many-target to one-source word alignment is possible by hypothesizing the same source
  position at different decoding steps. An example of such a case is shown in the hypothesis
  “Thanks a lot </s>” in Figure 5.3.

• There are no unaligned target words. All target words are generated using some source
  position.
5.6 Alignment-Based Neural Machine Translation

![Search Graph Example]

Figure 5.3: An excerpt of a search graph in alignment-based neural machine translation to translate the German sentence “Vielen Dank </s>”. The first entry of each node shows the coverage vector which we can keep track of in alignment-based neural machine translation. The second entry contains the word translation, followed by the position being translated. The last entry is the partial hypothesis score. Grey nodes are inactive nodes pruned away from the beam. Green nodes are terminal nodes that constitute full translation. A beam size of 3 is assumed. In the first decoding step, the word “Thanks” occurs twice in the beam, each time translated using a different alignment position. In this example, “Thanks a lot </s>” is the final translation with the best score.

- There can be unaligned source words. This is because we do not enforce full coverage of the source sentence.
- It is not possible to generate one-target to many-source translation. This algorithm has exactly one target source position per target word. While it is possible in principle to use models that capture this type of alignment, this increases the computational effort during search. If a maximum of $N$ source positions is to be allowed per target word, each decoding step will require hypothesizing $\sum_{n=1}^{N} \frac{J^n}{(J-n)!}$ positions. In our case $N = 1$, and as a result, a maximum of $J$ positions are hypothesized each decoding step.

5.6.4 Pruning

Since alignment-based search has an extra loop over source positions, this increases the computational effort by a factor of $J$. In order to reduce it, we introduce the following constraints:

- **Maximum coverage**: We introduce a limit on how many times a source position can be hypothesized per sentence. We set the maximum to 4 in our experiments.
1: function ADVANCE(hyps, prevAlignments, scores)
2:     localAlignScores ← ALIGNMENT DISTRIBUTION(hyps, prevAlignments)  \hspace{1cm} \triangleright \text{hyps batched}
3:     activePos ← ∅
4:     for source position $j = 1, \ldots, J$ do
5:         for beam entry $b = 1, \ldots, \text{beamSize}$ do
6:             if $\text{localAlignScores}[b, j] > \text{threshold}$ then
7:                 activePos.Append($j$)
8:             end if
9:         end for
10:     end for
11:     if activePos = ∅ then
12:         activePos ← \{1, \ldots, J\}
13:     end if
14:     for source position $j \in \text{activePos}$ do
15:         ($\text{hyps}[j], \text{localScores}[j]) ← \text{LEXICAL DISTRIBUTION}(\text{hyps}, j)$  \hspace{1cm} \triangleright \text{hyps batched}
16:         $\Delta = j - \text{prevAlignment}$
17:         $\text{scores}[j] ← \text{scores}[j] + \lambda \text{localScores}[j] + (1 - \lambda) \text{localAlignScores}[\Delta]$
18:     end for
19:     (hyps, alignments, scores) ← \text{GETBEST}(\text{hyps, alignments, scores, beamSize})
20:     return (hyps, alignments, scores)
21: end function

Algorithm 4: Threshold-based alignment pruning. The highlighted parts are due to the introduction of pruning to the function ADVANCE in Algorithm 3.

- **Diagonal constraint:** We use a window of width $2m + 1$ around the diagonal and only hypothesize source positions within this window. At target step $i$, we hypothesize the source positions $j \in \{i - m, \ldots, i + m\}$. $m$ is usually set to a large enough value, e.g. $m = 20$. For long source sentences, the complexity of search becomes constant in terms of the source sentence length.

- **Class-based lexical pruning:** While using a class-factored output layer in training reduces the computational effort due to the reduced softmax layer computation, this is not the case in decoding. As the word class is unknown, the full class and word output layers need to be computed, which is an overhead compared to not using word classes. To benefit from the class factorization in decoding, [Alkhouli & Bretschner+ 16] propose to prune the lexical hypothesis by limiting the number of hypothesized classes. In this case, the full class layer needs to be computed, but only the word layer parts corresponding to the high-scoring classes are evaluated.

- **Alignment pruning:** To reduce the number of hypothesized source positions further, [Alkhouli & Bretschner+ 18] suggest to prune the source positions that have bad alignment model scores. For each decoding step, the alignment model is evaluated first. The lexical model is only evaluated at the source positions that have high scores according to the alignment model. Alignment points that do not meet a pre-determined threshold are pruned away. Algorithm 4 introduces the changes to the function ADVANCE from Algorithm 3 to include the pruning changes. Since computing the lexical model is done batch-wise for all beam entries, an alignment position is pruned away only if it does not exceed the threshold for each beam entry. The alignment position will survive pruning if it exceeds the threshold.
for at least one of the beam entries. While this is not ideal, we keep batching because it is significantly faster than non-batched evaluation. We found that maintaining batch computation and using pruning as described does indeed improve speed. If none of the alignment positions survive pruning, the algorithm falls back to using all alignment positions (line 13).

5.6.5 Alignment Extraction

One of the options when constructing the neural network vocabulary is to limit it to the most frequent words and to map everything else to a special unknown token. Once an unknown word is generated, it is replaced by the source word used to generate it. It is also possible to use special category types to replace numbers, dates and other strings. Using the same copying mechanism, these values can be copied from the source to the target side. In this section, we discuss how to extract the alignment between target and source words online during decoding, where only the target history is assumed given. We describe how to extract the alignment for standard and alignment-based neural machine translation.

Since there is no explicit alignment information involved in standard neural machine translation, the attention weights can be used to extract the alignment. A simple approach is to align each target word to the source word that has the maximum attention weight

\[ j(i) = \arg \max_{j \in \{1...J\}} \{ \alpha_{i,j} \}, \]

where \( \alpha_{i,j} \) is the attention weight at source position \( j \) when generating the target word at position \( i \). In the case of multi-layer multi-head source-to-target attention, as in transformer networks, the maximum cumulative attention over layers and heads can be considered

\[ j(i) = \arg \max_{j \in \{1...J\}} \left\{ \sum_{l=1}^{L} \sum_{n=1}^{N} \alpha^{(l)}_{i,j,n} \right\}. \]

In the alignment-based approach, there is an alignment underlying the translation hypothesis. This alignment can be used, or alternatively, we can mix the attention and alignment information to get the final alignment. In the case of alignment-biased RNN attention, the following equation is used to extract: the alignment

\[ j(i, j') = \arg \max_{j \in \{1...J\}} \{ \alpha_{i,j,j'} \}, \quad (5.11) \]

where \( j(i, j') \) denotes the extracted alignment at target step \( i \) given a hypothesized source position \( j' \). \( \alpha_{i,j,j'} \) is computed using the energy described in Equation 4.47, where \( b_i \) is replaced by the hypothesized position \( j' \)

\[ r_{i,j,j'} = v^T \tanh(A_4 h_{j'} + A_5 s_{i-1} + a + \delta_{j,j'} b), \]

and \( \delta(\hat{j}, j') \) is the Kronecker delta

\[ \delta(\hat{j}, j') = \begin{cases} 1, & \text{if } j' = \hat{j}, \\ 0, & \text{otherwise}. \end{cases} \]

This formulation corresponds to extracting the maximum from alignment-biased attention weight. It biases the attention weights using the hypothesized position \( j' \).
We can also extract the alignment from the alignment-assisted transformer model as follows [Alkhouli & Bretschner+ 18]:

\[
    j(i, j') = \text{argmax}_{j \in \{1...J\}} \left\{ \sum_{l=1}^{L} \left( \sum_{n=1}^{N} \alpha_{i,l,n}^{(l)} + \delta_{j,j'} \right) \right\}.
\] (5.12)

Equation 5.12 biases the attention weights towards the hypothesized position \( j' \); therefore, if the attention weights are high enough, the attention sum will dominate. If, on the other hand, the attention weights have low value, the hypothesized source position will dominate. This allows to mix both information when choosing the final alignment. The attention weights can override the biased position if they are high enough.

An alternative for alignment-based models is to ignore the attention components and to use the hypothesized position \( j' \) directly as alignment. We compare both approaches in our experiments.

5.7 Experimental Evaluation

We carry out translation experiments using Jane, which is the RWTH Aachen University’s open-source toolkit for statistical machine translation. It is described in detail in [Vilar & Stein+ 10, Stein & Vilar+ 11, Wuebker & Huck+ 12]. We extend it to include standard and alignment-based neural machine translation decoding. The toolkit is mostly written in C++. The models used in Jane are trained using an extension of rwthlm [Sundermeyer & Schlüter+ 14] that includes feedforward, RNN attention, and alignment-biased attention lexical models. rwthlm also includes feedforward and RNN alignment models. In addition, we train RNN and self-attentive lexical and alignment models in Sockeye [Hieber & Domhan+ 17]. We extend it to include alignment-biased RNN attention and alignment-assisted transformer lexical models, as well as RNN and self-attentive alignment models. For decoding, we implement alignment-based search in Sockeye.

5.7.1 Setup

Table 5.5 shows a description of the neural systems we compare in our experiments. We carry out experiments on three translation tasks. We report the system configuration for each language pair individually. The corpora statistics for the WMT 2016 English→Romanian shared translation task are given in Table A.8. We limit the vocabulary of each of the source and target corpora to the 50k most frequent source and target words, respectively. We replace categories by special tokens in training, and replace these tokens after decoding in a post-processing step with the value that occurs in the source side. In the case of RNN attention-based and transformer models, we use the method described in Section 5.6.5 to determine the positions of the source words to be carried over to the target side. For purely alignment-based systems that lack the attention mechanism, we use the underlying alignment that generated the hypothesis to map the categories. Handling unknown words is done similarly, with a special unknown token that replaces infrequent word in the training corpus. The token is replaced by the aligned source word after decoding.

The phrase-based baseline uses 2 IBM1 and 2 phrasal models in both directions, 6 hierarchical reordering features, a language model, a bilingual language model, a word-class language model, in addition to the distortion penalty, the enhanced-low frequency, and the word and phrase penalty features.

All RNN models use 1 bidirectional encoder layer and 2 decoder layers. One of the decoder layers is located before the attention layer, and the other is located after it. The embedding size is 620. The recurrent layers are LSTM layers of size 1000, and the vocabulary consists of the most frequent 50k source and 50k target words. We apply dropout with a rate of 0.3. The transformer models use a stack of 6 encoder and 6 decoder layers. The feedforward layers of the transformer
Table 5.5: The description of the neural machine translation systems used in our experiments. The systems may consist of one lexical model or a combination of a lexical and an alignment model. The lexical models can either have feedforward (Section 4.5.1), recurrent (Sections 4.6.2, 4.5.6, 4.7.1) or self-attentive layers (Section 4.7.2). We also highlight whether the lexical model has a single attention layer (Sections 4.6.2, 4.7.1), multi-head attention layers (Sections 4.6.3, 4.7.2) or no attention layers (Sections 4.5.1, 4.5.6). The alignment models can have feedforward (Section 4.5.2), recurrent (Section 4.5.7) or self-attentive layers (Section 4.7.3). Systems marked in **bold face** are contributions of papers published towards this dissertation.

<table>
<thead>
<tr>
<th>system</th>
<th>models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>lexical</td>
</tr>
<tr>
<td>FFNN alignment</td>
<td>feedforward</td>
</tr>
<tr>
<td>RNN attention baseline + alignment</td>
<td>recurrent</td>
</tr>
<tr>
<td>RNN alignment</td>
<td>-</td>
</tr>
<tr>
<td>transformer baseline + alignment</td>
<td>self-attention</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

models are of size 2048. The size of all transformer layers and embeddings is set to 512. We apply dropout on all components with a rate of 0.2. The output label smoothing is set to 0.1. The models are trained using the Adam optimizer, with the initial learning rate set to 0.0003. The learning rate is halved if the validation perplexity does not improve for 3 consequent checkpoints.

We also perform experiments on the BOLT Chinese→English discussion forum task. Table A.4 shows the word-level statistics, and Table A.3 shows the BPE subword-level statistics. The phrase-based baseline is described in Section 6.7.1. The transformer models are similar to the ones used for the WMT English→Romanian task. The RNN models are also similar except for the encoder which is a stack of one bidirectional and 3 unidirectional LSTM layers to leverage the larger amount of training data. The transformer model training uses an initial learning rate of 0.0001 and reduces it by a factor of 0.7 if the validation perplexity does not improve for 3 consequent checkpoints.

We also include results for the WMT 2017 German→English shared translation task. The word- and subword-level corpus statistics are shown in Tables A.9 and A.10, respectively. The phrase-based baseline uses 2 IBM1 and 2 phrasal models in both directions, 6 hierarchical reordering features, and a language model, in addition to the distortion penalty, the enhanced-low frequency, and the word and phrase penalty features. The transformer and RNN models have the same configuration used for the BOLT Chinese→English task.

All alignment models have an output layer covering source jumps of width 100, in both left and right directions. The alignment-based systems use a diagonal window of width 40 for the WMT English→Romanian, and WMT German→English tasks, which means alignment points are only hypothesized if they are within 20 positions from the current target position. We use a window size of 60 for the BOLT Chinese→English task, making sure that the window size does not affect translation quality. The motivation for using a window is to speed up inference. Additionally, alignment pruning is applied for certain experiments. These experiments will be specifically highlighted. The maximum coverage per source position is set to 4 for alignment-based systems.
5.7.2 Alignment vs. Attention Systems

Table 5.6 shows results for experiments on the WMT 2016 English→Romanian translation task. This task is relatively small, containing about 15M running words. We compare between three main system categories: the phrase-based, the RNN and the transformer systems. We make the following observations:

- The phrase-based system is close in performance to the RNN attention baseline, but the transformer baseline outperforms both systems by quite a large margin. In general, we observe that it is challenging to train RNN attention systems to achieve relatively good performance on this task. We note that we use vocabularies limited to the most frequent 50k words. The OOV ratio is high (cf. Table A.8). In separate experiments, we tried training using BPE subwords, but we also observed comparable performance to the word-based RNN attention baseline system. It is possible that due to the limited amount of training data, the RNN decoder is not able to learn proper language modeling for the Romanian target language, which has complex morphology. In comparison, the phrase-based system uses three separate language models leveraging monolingual data, which is orders of magnitude more than the bilingual data. In contrast, the transformer baseline system is trained without much hyper-parameter tuning, but it is still able to outperform the other baselines. The transformer model leverages 48 attention heads in comparison to the single-head attention used in the RNN attention baseline. It also uses feedforward layers with residual connections, dropout, and layer normalization. These can be factors in stabilizing training even under limited-resource conditions.

- Adding alignment bias to the single-head attention component (line 3) improves the system. The alignment information is provided as an additional source of information during training, which turns out to be especially helpful for this small task.

- The pure alignment-based RNN system (line 4) outperforms the RNN attention baseline significantly. It is on par with the alignment-biased RNN attention system (line 3) in terms of BLEU but has worse TER scores. Using RNN attention combined with alignment is better than using pure attention or pure alignment.

- The alignment-based RNN systems trained on the small bilingual corpus (lines 3,4) significantly outperform the phrase-based baseline trained on additional monolingual data.

- Adding the extra alignment information as an additional attention head (line 6) to the transformer baseline does not change the performance of the system. It is harder to improve on this system which already has high performance. We will show, however, that the alignment-assisted transformer is better at generating word alignment hypotheses, which can be beneficial in certain scenarios.

To explore the performance of alignment-based systems on larger tasks, we experimented with The BOLT Chinese→English task. The results are shown in Table 5.7. We note the following observations:

- The phrase-based system underperforms all neural systems despite including two language models leveraging monolingual data. The transformer systems (lines 5–6) outperform the RNN systems (lines 2–4). This is consistent with our observations on the English→Romanian task. One possible reason for this are the multiple attention heads used by the transformer model, which can help capture the complex alignment of the Chinese-English language pair better than the single head of the RNN attention baseline.
Table 5.6: WMT 2016 English→Romanian results for the phrase-based baseline, the RNN attention baseline, and the transformer baseline. The baselines are compared to the RNN attention system with alignment bias, the RNN alignment system without attention, and the alignment-assisted transformer.

<table>
<thead>
<tr>
<th>#</th>
<th>system</th>
<th>params</th>
<th>newdev2016</th>
<th>newtest2016</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>BLEU [%]</td>
<td>TER [%]</td>
</tr>
<tr>
<td>1</td>
<td>phrase-based baseline</td>
<td>-</td>
<td>24.3</td>
<td>60.3</td>
</tr>
<tr>
<td>2</td>
<td>RNN attention baseline</td>
<td>147M</td>
<td>24.2</td>
<td>59.9</td>
</tr>
<tr>
<td>3</td>
<td>+ alignment</td>
<td>230M</td>
<td>25.6</td>
<td>56.5</td>
</tr>
<tr>
<td>4</td>
<td>RNN alignment</td>
<td>227M</td>
<td>25.6</td>
<td>57.6</td>
</tr>
<tr>
<td>5</td>
<td>transformer baseline</td>
<td>70M</td>
<td>27.4</td>
<td>55.5</td>
</tr>
<tr>
<td>6</td>
<td>+ alignment</td>
<td>161M</td>
<td>27.4</td>
<td>55.2</td>
</tr>
</tbody>
</table>

- Adding alignment bias (line 3) to the RNN attention baseline improves TER by up to 4.3%, and BLEU by up to 1.3%. When comparing the RNN attention baseline to the alignment-biased system, we noticed that the latter creates hypotheses with hypothesis-to-reference length ratio of 95.2–97.5% compared to the RNN attention baseline’s ratio of 98.9–101.7%. This difference can explain the large improvement in TER. In general, we observe that shorter hypotheses have better TER scores.

- The pure alignment-based RNN (line 4) is worse in performance compared to the alignment-biased RNN attention (line 3). Hence, using the combination of alignment and attention outperforms using pure alignment without attention. This is similar to our observation on the English→Romanian task. It is worth pointing that the pure alignment-based RNN is lagging behind the attention baseline by up to 0.9% BLEU. Given that these models do not use an attention component, this difference is less than anticipated. This may indicate that the RNN attention baseline is not sufficiently able to model the complex alignment dependencies of the Chinese-English language pair.

- Adding the extra alignment information (line 6) to the transformer baseline improves TER significantly on all data sets. This is also justified by the length ratio difference. The alignment-based transformer system has a ratio less by 1.2–1.8% absolute compared to the transformer baseline. The difference in BLEU is smaller with up to 0.4% improvement on P1R6-dev, and 0.3% degradation on DEV12-dev. We inspected the hypothesis-to-reference length ratio and found that the hypotheses on average are shorter than the reference on DEV12-dev compared to P1R6-dev. A closer look revealed that DEV12-dev has more long sentences, and the alignment-assisted system fails to match the reference length on long sentences. The average hypothesis-to-reference length ratio on DEV12-dev is 97.5% compared to 100.4% on P1R6-dev. Overall, the alignment-assisted transformer is comparable in performance to the transformer baseline in terms of BLEU and improved in terms of TER.

We also experimented with the WMT 2017 German→English task, which is a large task of 5.9M sentence pairs, and 144M English running words. The results are shown in Table 5.8. We observe the following:

- The performance of the phrase-based baseline is worse than all other neural network systems, and the transformer systems are the best performing among all systems. This is consistent with our observations on the previous two tasks.
5 Neural Machine Translation

Table 5.7: BOLT Chinese→English results for the phrase-based baseline, the RNN attention baseline, and the transformer baseline. The baselines are compared to the RNN attention system with alignment bias, the RNN alignment system without attention, and the alignment-assisted transformer.

<table>
<thead>
<tr>
<th>#</th>
<th>system</th>
<th>params</th>
<th>DEV12-tune</th>
<th>DEV12-dev</th>
<th>P1R6-dev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bleu [%]</td>
<td>Ter [%]</td>
<td>Bleu [%]</td>
</tr>
<tr>
<td>1</td>
<td>phrase-based baseline</td>
<td>-</td>
<td>20.1</td>
<td>66.0</td>
<td>18.2</td>
</tr>
<tr>
<td>2</td>
<td>RNN attention baseline</td>
<td>167M</td>
<td>22.1</td>
<td>65.0</td>
<td>20.8</td>
</tr>
<tr>
<td>3</td>
<td>+ alignment</td>
<td>277M</td>
<td>22.1</td>
<td>62.2</td>
<td>20.8</td>
</tr>
<tr>
<td>4</td>
<td>RNN alignment</td>
<td>275M</td>
<td>20.7</td>
<td>64.2</td>
<td>19.9</td>
</tr>
<tr>
<td>5</td>
<td>transformer baseline</td>
<td>120M</td>
<td>24.2</td>
<td>61.5</td>
<td>23.2</td>
</tr>
<tr>
<td>6</td>
<td>+ alignment</td>
<td>212M</td>
<td>24.3</td>
<td>60.5</td>
<td>22.9</td>
</tr>
</tbody>
</table>

- Biasing the RNN attention baseline with alignment (line 3) does not improve the attention baseline (line 2). There is even a slight degradation on the test set newstest2017. These performance results however do not reflect the internal alignment quality of the system. We analyze this further in Sections 5.7.11 and 5.7.12, and demonstrate that adding alignment significantly improves the alignment and therefore alignment-dependent tasks such as dictionary suggestions.

- The pure alignment-based system is worse than the RNN attention baseline. We note that the RNN alignment system performs worse on this task compared to the two previous tasks. The German→English language pair includes complex alignment cases, such as long-range reordering, and one possible reason for the relatively good attention baseline performance is that a sufficient amount of training data is available to train the attention component to model such complex reordering. Another indication for this emerges when comparing the attention baseline and the combination of attention and alignment (line 3), where we observe no improvement in using the alignment in this case, contrary to what we observe on the previous language pairs.

- Similar to the RNN systems, adding alignment information (line 6) to the transformer baseline (line 5) slightly degrades its performance. However, we will demonstrate that using the alignment information allows for a better alignment extraction from the system which can be helpful in some alignment-based tasks.

In summary, we consistently observe that the transformer model outperforms the RNN and phrase-based systems. Adding alignment to the English→Romanian and Chinese→English tasks significantly improved the RNN attention baseline systems, but this could not be observed on the German→English task. In separate experiments, we observe that the alignment error rate of the German→English system is higher for alignment-based systems compared to that of the corresponding Chinese→English systems (as will be presented in Table 5.17). This can be specific to the German→English language pair that has long-range reordering. Still, alignment-biased RNN attention systems have competitive performance in comparison to the RNN attention baseline variants. We also observed that adding the alignment information to the transformer system does maintain its performance, albeit with negligible degradation in the English→German system. We will demonstrate that alignment-based RNN and transformer systems have an advantage in tasks that require alignment extraction over the pure attention-based systems.
5.7 Experimental Evaluation

Table 5.8: WMT 2017 German→English results for the phrase-based baseline, the RNN attention baseline, and the transformer baseline. The baselines are compared to the RNN attention system with alignment bias, the RNN alignment system without attention, and the alignment-assisted transformer.

<table>
<thead>
<tr>
<th>#</th>
<th>system</th>
<th>params</th>
<th>newstest2015</th>
<th>newstest2017</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bleu [%]</td>
<td>Ter [%]</td>
</tr>
<tr>
<td>1</td>
<td>phrase-based baseline</td>
<td>-</td>
<td>29.9</td>
<td>54.2</td>
</tr>
<tr>
<td>2</td>
<td>RNN attention baseline</td>
<td>171M</td>
<td>31.2</td>
<td>50.8</td>
</tr>
<tr>
<td>3</td>
<td>+ alignment</td>
<td>277M</td>
<td>30.7</td>
<td>51.5</td>
</tr>
<tr>
<td>4</td>
<td>RNN alignment</td>
<td>275M</td>
<td>29.4</td>
<td>51.9</td>
</tr>
<tr>
<td>5</td>
<td>transformer baseline</td>
<td>70M</td>
<td>33.2</td>
<td>48.7</td>
</tr>
<tr>
<td>6</td>
<td>+ alignment</td>
<td>161M</td>
<td>33.0</td>
<td>49.1</td>
</tr>
</tbody>
</table>

Table 5.9: A comparison between RNN alignment systems using two types of vocabulary: top 50k source and target words, and byte-pair encoded subwords. The subword vocabularies are computed using 50k merge operations trained jointly on the source and target corpora, except for the Chinese→English system, where the source and target languages use completely different characters; therefore, we trained the BPE operations separately on the Chinese and English corpora. We limit the subword vocabularies to a maximum of 50k subwords.

<table>
<thead>
<tr>
<th>#</th>
<th>Output layer type</th>
<th>WMT En→Ro newstest2016</th>
<th>WMT Zh→En DEV12-dev</th>
<th>WMT De→En newstest2017</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Bleu [%]</td>
<td>Ter [%]</td>
<td>Bleu [%]</td>
</tr>
<tr>
<td>1</td>
<td>50k words</td>
<td>26.7</td>
<td>56.1</td>
<td>18.7</td>
</tr>
<tr>
<td>2</td>
<td>BPE units</td>
<td>24.2</td>
<td>58.3</td>
<td>19.9</td>
</tr>
</tbody>
</table>

5.7.3 Word vs. Byte-Pair-Encoded Subword Vocabulary

In Table 5.9, we compare between using source and target vocabularies limited to the top frequent 50k words, and byte-pair encoded subword vocabularies. We observe that in the case of the smaller English→Romanian task, using the top 50k words is better than using subwords, while for the larger German→English and Chinese→English tasks, using subwords has a clear advantage over limiting the vocabulary to the frequent 50k words. Limiting the vocabulary results in an increase in the OOV rate. Still the results indicate that for the smaller task, using words outweighs using subwords. Based on this, we use words for the English→Romanian system, and BPE subwords for the German→English and Chinese→English systems.

5.7.4 Feedforward vs. Recurrent Alignment Systems

Next, we compare feedforward and recurrent alignment-based systems that do not have any attention component. We perform experiments on the same three language pairs discussed in the previous section. Table 5.10 shows the WMT 2016 English→Romanian results, Table 5.11 shows the BOLT Chinese→English results, and Table 5.12 shows the WMT 2017 German→English results. We compare between pure alignment-based feedforward systems consisting of a feedforward lexical and feedforward alignment models, and pure alignment-based bidirectional RNN systems having bidirectional RNN lexical and alignment models.
For the English→Romanian task, the feedforward lexical and alignment models have a 1000-node layer followed by a 500-node layer each. We apply dropout on both layers with a rate of 0.1 on this particular task of small data, as it improves the models significantly. The model input is a window of 9 source words centered at the aligned source position, and 5 target words. The embedding size is 100. The vocabulary is the most frequent 81k target words and 56k source words. The same vocabulary is used by the class-factored RNN lexical models. We use 200-node LSTM layers for the lexical and alignment models of the class-factored RNN systems. Each of the models consist of a bidirectional encoder, a unidirectional LSTM layer on the target words, and another LSTM after the encoder and target LSTM layer outputs are combined. The word embedding size is 200. The vocabulary of the German→English task is 188k source words and 131k target words, and the models use 200-node LSTMs and word embeddings of size 200. For the Chinese→English systems, we use 128k source words, 169k target words, 350-node LSTMs and embeddings of size 350. We also include larger RNN networks of 1000-node LSTM layers, and embeddings of size 620. For these larger models, we keep the same number of layers for the English→Romanian system due to the limited amount of training data. For the German→English and Chinese→English systems, we use a deeper encoder for the large models consisting of 1 bidirectional and 3 stacked unidirectional encoder layers on top of it.

The class assignment used in the class-factored output layer is done offline using maximum-likelihood training with the bigram assumption using the exchange algorithm [Kneser & Ney 91]. When a class-factored output layer is used, the word layer is only evaluated for the 3 top scoring classes. The class-factored lexical models use 2k classes, 1k of which are dedicated to the most frequent 1k words, and the rest are shared among the remaining target words. All alignment models have an output layer covering source jumps of width 100, in both left and right directions. The models of the class-factored systems are trained using SGD with annealing. The learning rate is halved when the validation perplexity of the current checkpoint does not improve compared to the last checkpoint. In this case, the effect of the checkpoint that did not improve the perplexity is discarded, and training is restarted with the updated learning rate using the previous checkpoint. All systems with a class-factored output layer use class-based lexical pruning using the top 3 classes during decoding. The maximum coverage per source position is set to 4 for alignment-based systems.

We either use class-factored output layer or a full output layer limited to the most frequent 50k (sub)words. In Tables 5.10–5.12, lines 1–2 use SGD with newbob scheduling, where the learning rate is scaled by 0.5 if the validation perplexity does not improve. Lines 3–4 use Adam for optimization, which has dynamic per-parameter learning rate scheduling. We refer the reader to [Bahar & Alkhouli + 17] for an empirical comparison between optimization algorithms. We choose these settings because we observe little difference between Adam and annealed SGD when training models with a small hidden layer size, but we prefer to train large RNN models with Adam, because Adam has a dedicated learning rate per parameter, while SGD only has a global learning rate. We note the following observations:

- RNN systems outperform feedforward systems on all tasks (lines 1–2). There are several reasons why RNN systems perform better. They use a bidirectional encoder that encodes the full source sentence, compared to a window used by the feedforward systems. Second, they are conditioned on the full alignment path, while the feedforward model computation only requires the current alignment point. Third the RNN systems use the full target history, while the feedforward models use the most recent 5 target words. Because the RNN systems make use of all available source, alignment and target information, they outperform the feedforward systems. Note that there is little or no difference when comparing the overall number of parameters of the two systems.
Table 5.10: A comparison between alignment-based feedforward and recurrent systems without attention on the WMT 2016 English→Romanian shared translation task.

<table>
<thead>
<tr>
<th>#</th>
<th>output</th>
<th>system</th>
<th>params</th>
<th>newdev2016_1 BLEU [%]</th>
<th>newdev2016_1 TER [%]</th>
<th>newtest2016 BLEU [%]</th>
<th>newtest2016 TER [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>classes</td>
<td>FFNN alignment</td>
<td>74M</td>
<td>22.4</td>
<td>61.4</td>
<td>23.3</td>
<td>60.4</td>
</tr>
<tr>
<td>2</td>
<td>RNN alignment</td>
<td>74M</td>
<td>23.7</td>
<td>59.6</td>
<td>24.5</td>
<td>58.6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>50k words</td>
<td>RNN alignment</td>
<td>52M</td>
<td>22.3</td>
<td>60.2</td>
<td>23.7</td>
<td>58.6</td>
</tr>
<tr>
<td>4</td>
<td>50k words</td>
<td>RNN alignment</td>
<td>227M</td>
<td>25.6</td>
<td>57.6</td>
<td>26.7</td>
<td>56.1</td>
</tr>
</tbody>
</table>

Table 5.11: A comparison between alignment-based feedforward and recurrent systems without attention on the BOLT Chinese→English discussion forum task. Rows 3–4 are run using byte-pair encoding subwords using 50K merge operations.

<table>
<thead>
<tr>
<th>#</th>
<th>output</th>
<th>system</th>
<th>params</th>
<th>DEV12-Tune BLEU [%]</th>
<th>DEV12-dev BLEU [%]</th>
<th>P1R6-dev BLEU [%]</th>
<th>P1R6-dev TER [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>classes</td>
<td>FFNN alignment</td>
<td>251M</td>
<td>15.1</td>
<td>70.6</td>
<td>13.8</td>
<td>70.6</td>
</tr>
<tr>
<td>2</td>
<td>RNN alignment</td>
<td>258M</td>
<td>16.7</td>
<td>69.1</td>
<td>16.0</td>
<td>69.5</td>
<td>15.7</td>
</tr>
<tr>
<td>3</td>
<td>BPE units</td>
<td>RNN alignment</td>
<td>94M</td>
<td>18.6</td>
<td>65.3</td>
<td>16.5</td>
<td>66.6</td>
</tr>
<tr>
<td>4</td>
<td>BPE units</td>
<td>RNN alignment</td>
<td>275M</td>
<td>20.7</td>
<td>64.2</td>
<td>19.9</td>
<td>65.1</td>
</tr>
</tbody>
</table>

- Switching the RNN system from a class-factored output layer to a 50k output layer has an impact on the performance (lines 2–3). On the small English→Romanian task, the 50k word output layer degrades the performance. We note this reduces the vocabulary size, and hence increases the number of out-of-vocabulary (OOV) words (cf. Table A.8). This increase in the OOV rate can justify the degradation. As demonstrated in Table 5.9, using byte-pair encoded vocabulary for this task yielded worse results than using the most frequent 50k words. On the other hand, we use byte-pair encoded subwords for the German→English and Chinese→English systems. For these larger tasks, switching from the class-factored to the BPE output layer results in a significant improvement. The OOV rate is reduced to 0.0% on the target side when using subwords.

- The best results are achieved when increasing the LSTM layer size to 1000 nodes per layer, and the source and target embedding dimension to 620 (line 4). We note that training larger models with Adam is feasible, but it is harder to train good performing large models when using SGD with annealing. Adam has a dedicated learning rate per parameter, while SGD only has a global learning rate.

We compare recurrent and feedforward lexical and alignment models in Table 5.13. We observe that replacing the feedforward alignment model (line 1) by the bidirectional RNN alignment model (line 2) improves the system. We even observe stronger improvement when doing the replacement in a system using bidirectional lexical model (lines 3–4). The feedforward alignment model has a window over the source; therefore, predicting a jump to a word outside the window boundaries will exclude the lexical context around the position to be jumped to. In contrast, the bidirectional RNN states are computed using the full context of the source sentence, making the jump prediction conditioned on more lexical source information. We also observe improvement when replacing the feedforward lexical models (lines 1 and 3) by the bidirectional RNN lexical
Table 5.12: A comparison between alignment-based feedforward and recurrent systems without attention on the WMT 2017 German→English shared translation task. Rows 3–4 are run using byte-pair encoding subwords using 50K merge operations.

<table>
<thead>
<tr>
<th>#</th>
<th>output</th>
<th>system</th>
<th>params</th>
<th>newtest2015 BLEU [%]</th>
<th>newtest2017 BLEU [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>classes</td>
<td>FFNN alignment</td>
<td>133M</td>
<td>23.9</td>
<td>24.2</td>
</tr>
<tr>
<td>2</td>
<td>classes</td>
<td>RNN alignment</td>
<td>153M</td>
<td>26.5</td>
<td>26.8</td>
</tr>
<tr>
<td>3</td>
<td>BPE units</td>
<td>RNN alignment</td>
<td>48M</td>
<td>27.0</td>
<td>27.6</td>
</tr>
<tr>
<td>4</td>
<td>BPE units</td>
<td>RNN alignment</td>
<td>275M</td>
<td>28.8</td>
<td>29.2</td>
</tr>
</tbody>
</table>

Table 5.13: BOLT Chinese→English results highlighting the effect of feedforward and recurrent lexical and alignment models.

<table>
<thead>
<tr>
<th>#</th>
<th>lexical model</th>
<th>alignment model</th>
<th>DEV12-tune BLEU [%]</th>
<th>DEV12-dev BLEU [%]</th>
<th>P1R6-dev BLEU [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FFNN</td>
<td>FFNN</td>
<td>15.1 70.6</td>
<td>14.5 70.7</td>
<td>13.8 70.6</td>
</tr>
<tr>
<td>2</td>
<td>FFNN</td>
<td>RNN</td>
<td>15.8 70.6</td>
<td>14.7 71.1</td>
<td>14.5 70.7</td>
</tr>
<tr>
<td>3</td>
<td>RNN</td>
<td>FFNN</td>
<td>16.0 69.3</td>
<td>15.2 69.6</td>
<td>14.6 69.5</td>
</tr>
<tr>
<td>4</td>
<td>RNN</td>
<td>RNN</td>
<td>16.7 69.1</td>
<td>16.0 69.5</td>
<td>15.7 68.7</td>
</tr>
</tbody>
</table>

model (lines 2 and 4), which can also be explained due to the more expressive power of the bidirectional RNN model, which captures unbounded source and target context. The best system is achieved when using a combination of bidirectional lexical and alignment models.

### 5.7.5 Bidirectional Recurrent Lexical Model Variants

In Table 5.14 we compare different alternatives for choosing the source representation used in Equation 4.35. This equation indicates where the source and target information are combined in the model. The systems are RNN alignment systems without attention. We observe that including the current bidirectional state $h_b_i$ is important to have the best performance. Replacing it by the previously aligned state $h_b_{i-1}$ results in a severe degradation. This is because at step $i$, the word at position $b_i$ is being translated; therefore, it is essential to include a representation of that position. Including the source embedding $x_b_i$ of the source word at $b_i$ (line 3) does recover most of the lost performance, but it does not reach the level of using the bidirectional representation $h_b_i$. The bidirectional representation is context-dependent since it is computed using all future and past source words; therefore, it is a more informed representation than the word embedding. We do not observe an improvement when including both $h_b_i$ and $h_b_{i-1}$ at the same time (line 4); therefore, we use the simpler variant described in line 1 in our experiments.

### 5.7.6 Alignment-Biased Attention

In this section we analyze the effect of biasing the attention component using external alignment information (cf. Equation 4.47). All our alignment-biased attention systems are trained by enabling the alignment information for 50% of the training batches. We do this to avoid overfitting the alignment, and to allow learning attention parameters that can still attend to the source
Table 5.14: Translation results on the WMT 2016 English→Romanian task for different bidirectional recurrent lexical model structures. The source representation is used in Equation 4.35. $h_b_i$ is the bidirectional source state aligned to the target position $i$, similarly, $h_{b_{i-1}}$ is the bidirectional state aligned to the previous target position $i-1$. $x_{b_i}$ is the embedding of the source word at position $b_i$. Lines 3–4 include multiple source representations by aggregation.

<table>
<thead>
<tr>
<th>#</th>
<th>source representation</th>
<th>newsdev2016</th>
<th>newsnewstest2016</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Bleu [%]</td>
<td>Ter [%]</td>
</tr>
<tr>
<td>1</td>
<td>$h_b_i$</td>
<td>23.7</td>
<td>59.6</td>
</tr>
<tr>
<td>2</td>
<td>$h_{b_{i-1}}$</td>
<td>19.8</td>
<td>66.5</td>
</tr>
<tr>
<td>3</td>
<td>$h_{b_{i-1}} + x_{b_i}$</td>
<td>22.8</td>
<td>61.2</td>
</tr>
<tr>
<td>4</td>
<td>$h_{b_{i-1}} + h_{b_i}$</td>
<td>23.8</td>
<td>60.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24.5</td>
<td>58.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20.5</td>
<td>65.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>23.5</td>
<td>60.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24.4</td>
<td>59.3</td>
</tr>
</tbody>
</table>

positions without the alignment information. Table 5.15 shows different configurations to measure the effect of blocking out the alignment information in training. All the systems in the table are trained using 200-node LSTM layers and 200-node attention layers, and they use a class-factored output layer. Line 1 shows the RNN attention baseline. Adding alignment information to all batches during training the lexical model is shown in line 2. This system performs decoding using the alignment-based decoder, including a bidirectional recurrent alignment model. Adding the alignment information improves the attention baseline. To check whether the attention component is working, we use the standard decoder including only the lexical model, which is trained using the alignment information. This configuration is shown in line 3. We observe that the system is not able to produce any meaningful translation at all, indicating the attention component is dysfunctional. This happens because the model depends completely on the external alignment information during training, and therefore does not learn useful parameters for the attention component. In line 4, we enable the alignment for 50% of the training batches only. This improves over the system that includes alignment all the time (line 2) on the WMT 2016 English→Romanian task. It does not improve over the WMT 2017 German→English system which has much more data and is less prone to overfitting. When checking whether the attention component is working by disabling the use of alignment during decoding (line 5), we find that the model recovers its baseline performance (line 1), although it is trained using alignment. This means that although the alignment information is used in training, it does not affect the training of the attention component. To benefit from both alignment and attention components, we use the alignment information 50% of the time during. We apply this to all alignment-biased systems.

### 5.7.7 Class-Factored Output Layer

Choosing a class-factored output layer is beneficial for speeding up training in comparison to using a standalone output layer of a large vocabulary. We propose class-based lexical pruning to also speed up decoding (cf. Section 5.6.4). Figure 5.4 shows the translation quality measured in BLEU and the decoding speed in words per second. Search is limited to evaluating the word layer parts that correspond to the top $k$ classes. The x-axis shows the number of top classes used in search. The baseline performance of computing all of the word layer yields 16.7 BLEU using all 2000 classes. On the other extreme, using the one-best class leads to a loss of 0.2% BLEU, while speeding up decoding from 0.01 to 0.29 words per second. In all of our experiments, we use the top 3 classes, which has the same translation quality as the non-pruned case, and has a decoding speed of 0.27 words per second. This means that decoding speed is 27 times faster than the non-pruned system, while maintaining translation quality.
Table 5.15: The effect of using the alignment information during training. We compare between including the alignment information for all training batches vs. including it 50% of the training time. We randomly choose whether or not to include the alignment information, and the decision is done on the batch level. This affects training only. When alignment is enabled during decoding, it is used all the time. Decoding without alignment is performed using the standard neural machine translation decoder that does not have a loop over alignment. No alignment input is provided to the lexical model in this case, and no alignment model is used either.

<table>
<thead>
<tr>
<th>#</th>
<th>system</th>
<th>WMT En→Ro newstest2016</th>
<th>WMT De→En newstest2017</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BLEU [%]</td>
<td>TER [%]</td>
</tr>
<tr>
<td>1</td>
<td>RNN attention baseline</td>
<td>23.1</td>
<td>60.6</td>
</tr>
<tr>
<td>2</td>
<td>+ alignment (train 100%)</td>
<td>23.7</td>
<td>59.2</td>
</tr>
<tr>
<td>3</td>
<td>+ decode without alignment</td>
<td>degenerate</td>
<td>degenerate</td>
</tr>
<tr>
<td>4</td>
<td>+ alignment (train 50%)</td>
<td>24.8</td>
<td>58.1</td>
</tr>
<tr>
<td>5</td>
<td>+ decode without alignment</td>
<td>23.1</td>
<td>60.6</td>
</tr>
</tbody>
</table>

Figure 5.4: Decoding speed and translation quality using top scoring classes in a class-factored output layer. The results are computed for the BOLT Chinese→English DEV12-tune development set.
5.7 Experimental Evaluation

5.7.8 Model Weights

There are two models in the log-linear combination of alignment-based systems. The log probabilities are weighted and summed up. The lexical model weight is $\lambda \in [0, 1]$ and the alignment model weight is $1 - \lambda$. We tuned $\lambda$ on the development set and selected the weight that resulted in the best BLEU then TER scores for each system. Figure 5.5 shows the translation quality for the development set newsdev2016 of the WMT 2016 English→Romanian task plotted against the lexical model weight $\lambda$. We observe that the system performance is robust against small changes in $\lambda$. In the figure, the best system is achieved when $\lambda = 0.8$.

5.7.9 Alignment Variants

Training the alignment-based systems discussed thus far relies on precomputed word alignment for the parallel training data. We explore the effect of using different alignment variants on translation quality. Table 5.16 shows the translation quality in BLEU and the time needed to create one alignment direction for the training data. The final alignment used for training is obtained by merging the source-to-target and target-to-source alignment directions. The comparison is between RNN attention systems using alignment bias. We observe that IBM4 alignment yields the best performance, but requires more training time. This is due to their complex dependencies and training strategy. The less complex first-order hidden Markov model (HMM) alignment is slightly worse in translation quality, but faster to generate. The IBM2 variant proposed by [Dyer & Chahuneau+ 13] has zero-order dependencies, and it is the fastest to compute. It loses 1.3% BLEU in comparison to the more complex IBM4 alignment system, while being 8 times faster. The last row is the RNN attention baseline which does not use any alignment. Comparing all three alignment-based systems to the attention baseline that uses no alignment shows an advantage for alignment-based systems on this task.

5.7.10 Alignment Pruning

Figures 5.6 and 5.7 show experiments using the alignment-assisted transformer system. We vary the pruning threshold which controls the number of alignment points at which the lexical model is evaluated during each decoding step. A high threshold results in more aggressive pruning. We
Table 5.16: A comparison between different alignment variants in terms of the time needed to compute them prior to training the neural networks, and the translation quality resulted from using them. The translation system is the RNN attention with alignment bias, which also uses a bidirectional recurrent alignment model. The results are for the WMT 2016 English→Romanian task which has 604k sentence pairs. We report the time needed to align the training data on a single-CPU machine.

<table>
<thead>
<tr>
<th>#</th>
<th>Alignment</th>
<th>Time</th>
<th>WMT En→Ro newstest2016 BLEU [%]</th>
<th>TER [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IBM4</td>
<td>2.5h</td>
<td>26.6</td>
<td>55.5</td>
</tr>
<tr>
<td>2</td>
<td>HMM</td>
<td>1.3h</td>
<td>25.8</td>
<td>56.3</td>
</tr>
<tr>
<td>3</td>
<td>IBM2 (fast align)</td>
<td>0.3h</td>
<td>25.3</td>
<td>56.4</td>
</tr>
<tr>
<td>4</td>
<td>none</td>
<td>-</td>
<td>24.7</td>
<td>58.9</td>
</tr>
</tbody>
</table>

Figure 5.6: Translation time in seconds and quality in BLEU vs. the pruning threshold. The results are computed for the newsdev2016 set of the WMT English→Romanian task. We report the time it took to translate the data set using a single GPU and batch size 1.

Figure 5.7: Translation time in seconds and quality in BLEU vs. the pruning threshold. The results are computed for the newsdev2016 set of the WMT English→Romanian task. We report the time it took to translate the data set using a single GPU and batch size 5.
plot BLEU and the time needed to translate the whole development set. Figure 5.6 corresponds to experiments run with batch size 1, and Figure 5.7 corresponds to batch size 5. We speed up translation by a factor of 2.7 and 1.8 for batch sizes 1 and 5, respectively, for a threshold of 0.15 without loss in translation quality. Higher threshold values result in more aggressive pruning and hence lead to a degradation in translation quality. It is interesting to note that at threshold 0.05 we achieve a speed up of 2.5 and 1.7 for batch sizes 1 and 5, respectively, implying that significant pruning happens at low threshold values. Translation requires more time at high threshold values, because we have more cases where no alignment points survive the threshold, in which case Algorithm 4 falls back to evaluating all source positions. Comparing batch sizes 1 and 5, we observe speed up in both cases, but the speed up is larger in the case of batch size 1. To explain this, we refer the reader to Algorithm 4. In the case of batching, an alignment point is considered active if its alignment score survives the threshold for any of the sentences in the batch. This results in more active alignment points for larger batches, and hence less speed up. On the other hand, pruning still improves speed in the case of batching, and overall, the combination of batching and pruning results in the best speed, where the development set is translated three times faster than the case of no batching at threshold 0.15.

5.7.11 Alignment Quality

Next, we investigate the alignment quality of the different systems included in our study. To this end, we use 504 manually aligned German-English sentence pairs provided by [Vilar & Popović+ 06] which is from the European parliament domain, and 900 manually aligned Chinese-English sentence pairs provided by [Liu & Sun 15]. We use the alignment error rate [Och & Ney 00] to measure the alignment quality. Let $A = \{(i, b_i) | b_i > 0\}$ denote the set of alignment points which we want to measure its quality. The alignment error rate (AER) is defined as follows:

$$AER = 1 - \frac{|A \cap S| + |A \cap P|}{|A| + |S|}, \quad (5.13)$$

where $S$ and $P$ denote the manual alignment sets corresponding to “sure” and “possible” alignment points.

The manual alignment is given on the word level, while our German→English and Chinese→English systems are trained on the subword level. Since we use large subword vocabularies, we expect most words to exist in the full form in the training corpus; therefore, we still use our subword-trained systems to compute word-level alignment for the test sets in this task. Since our training and decoding schemes for alignment-based systems assume each target word to be aligned exactly to one source word, we convert the manual alignment data to the same format. We apply the same alignment heuristics used on the training data to resolve alignment ambiguities in the manual alignment. In particular, unaligned target words are aligned according to the nearest aligned target word. In the case of multiple-aligned source words, we align the target word to the middle source word and ignore all other links. By doing this, we ensure consistency between the alignment generated by the neural systems and the manual alignment. In the standard RNN attention and transformer systems, we use the position having the maximum (accumulated) attention weight as alignment point, as described in Section 5.6.5. For alignment-based systems, we use the alignment path underlying the best translation, i.e. the attention weights are not directly considered when extracting the alignment.

Table 5.17 shows the alignment error rate computed for the different systems. We observe the following:

- As expected, adding alignment (line 2) to the attention baseline (line 1) improves the alignment quality significantly. Similarly, adding alignment (line 6) to the transformer baseline (line 4) results in significant improvements.
Table 5.17: Alignment error rate results for the German→English and Chinese→English systems.

<table>
<thead>
<tr>
<th>#</th>
<th>system</th>
<th>AER[%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>WMT De→En</td>
</tr>
<tr>
<td>1</td>
<td>RNN attention baseline</td>
<td>47.5</td>
</tr>
<tr>
<td>2</td>
<td>+ alignment</td>
<td>40.9</td>
</tr>
<tr>
<td>3</td>
<td>RNN alignment</td>
<td>31.7</td>
</tr>
<tr>
<td>4</td>
<td>transformer baseline</td>
<td>64.9</td>
</tr>
<tr>
<td>5</td>
<td>+ ignore sentence end</td>
<td>44.4</td>
</tr>
<tr>
<td>6</td>
<td>transformer + alignment</td>
<td>37.6</td>
</tr>
<tr>
<td>7</td>
<td>+ ignore sentence end</td>
<td>37.6</td>
</tr>
</tbody>
</table>

- The pure RNN alignment system (line 3) outperforms the alignment-biased RNN attention (line 2) on both language pairs, even though the alignment information is also included in the latter. This indicates that the use of the attention component leads to more alignment errors. Overall, the pure RNN alignment model has the most competitive results in terms of alignment quality in comparison to all other systems.

- The transformer baseline (line 4) performs poorly in comparison to all other systems. We noticed that the model focuses often times on the sentence end in the middle of translation. In line 5, we ignore the sentence end when choosing the alignment position, i.e., we compute the alignment as the maximum accumulated attention weight, maximizing over all source positions excluding the sentence end. This yields a significant improvement. This emphasizes the issue of incorrect sentence-end alignment. In comparison, the alignment-based transformer does not have this issue (line 6 vs. 7).

It is not surprising that explicit alignment modeling has superior performance to the RNN attention and transformer baselines. In Section 5.7.2, we observed that the alignment-based systems have at least similar translation performance or sometimes outperform their corresponding baselines. These results combined with the improved alignment error rates lead to the following question: how do the alignment-based systems compare to the baseline systems in alignment-based translation tasks? By alignment-based tasks we mean tasks that depend on the alignment information. One such task is the dictionary suggestions task, which we discuss next.

### 5.7.12 Dictionary Suggestions

Alignment-assisted transformer systems have comparable results to the baseline transformer system, but they have better alignment quality as shown in the previous section. In this section, we investigate whether the relatively good alignment quality of alignment-based systems can be useful for practical translation applications. We run comparison experiments that exploit the alignment information to bias translation using a user-provided dictionary. We refer to these experiments as dictionary suggestion experiments. The dictionary consists of source-target word pairs. The goal is to have the dictionary source words translated to the dictionary target words during sentence translation. We perform translation override online during decoding, by setting infinite costs to all but the desired target word. We determine the source word being translated using the attention weights. In the standard RNN attention and transformer systems, we use the position having the maximum (accumulated) attention weight as the alignment point, as described in Section 5.6.5. In alignment-based systems, we can use either the maximum attention weight as described in Section 5.6.5, or the hypothesized position ignoring attention weights. This task...
Table 5.18: Number of dictionary entries used in each system in Table 5.19, and the percentage of matching entries relative to number of words in the tokenized reference.

<table>
<thead>
<tr>
<th>#</th>
<th>system</th>
<th>WMT En→Ro</th>
<th>BOLT Zh→En</th>
<th>WMT De→En</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#</td>
<td>%</td>
<td>#</td>
</tr>
<tr>
<td>1</td>
<td>RNN attention baseline</td>
<td>2,571</td>
<td>10</td>
<td>3,805</td>
</tr>
<tr>
<td>2</td>
<td>RNN attention + alignment</td>
<td>2,454</td>
<td>9</td>
<td>3,770</td>
</tr>
<tr>
<td>3</td>
<td>RNN alignment</td>
<td>2,451</td>
<td>9</td>
<td>3,874</td>
</tr>
<tr>
<td>4</td>
<td>transformer baseline</td>
<td>2,389</td>
<td>9</td>
<td>3,533</td>
</tr>
<tr>
<td>5</td>
<td>transformer + alignment</td>
<td>2,389</td>
<td>9</td>
<td>3,611</td>
</tr>
</tbody>
</table>

Table 5.19: The effect of using the dictionary in constrained decoding. It is only meaningful to compare the systems with dictionaries to each other. Comparing a system using a dictionary to one that does not can be misleading, since the dictionaries are extracted given the dataset reference information, which is a form of cheating.

<table>
<thead>
<tr>
<th>#</th>
<th>system</th>
<th>WMT En→Ro</th>
<th>BOLT Zh→En</th>
<th>WMT De→En</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>newstest2015</td>
<td>Bleu [%]</td>
<td>Ter [%]</td>
<td>Bleu [%]</td>
</tr>
<tr>
<td>1</td>
<td>RNN attention baseline</td>
<td>24.2</td>
<td>59.9</td>
<td>22.1</td>
</tr>
<tr>
<td>2</td>
<td>+ dictionary</td>
<td>25.7</td>
<td>60.8</td>
<td>22.0</td>
</tr>
<tr>
<td>3</td>
<td>RNN attention + alignment</td>
<td>25.6</td>
<td>56.5</td>
<td>22.1</td>
</tr>
<tr>
<td>4</td>
<td>+ dictionary</td>
<td>27.9</td>
<td>55.4</td>
<td>24.4</td>
</tr>
<tr>
<td>5</td>
<td>RNN alignment</td>
<td>25.6</td>
<td>57.6</td>
<td>20.7</td>
</tr>
<tr>
<td>6</td>
<td>+ dictionary</td>
<td>28.2</td>
<td>56.3</td>
<td>22.7</td>
</tr>
<tr>
<td>7</td>
<td>transformer baseline</td>
<td>27.4</td>
<td>55.5</td>
<td>24.2</td>
</tr>
<tr>
<td>8</td>
<td>+ dictionary</td>
<td>29.6</td>
<td>55.3</td>
<td>25.6</td>
</tr>
<tr>
<td>9</td>
<td>transformer + alignment</td>
<td>27.4</td>
<td>55.2</td>
<td>24.3</td>
</tr>
<tr>
<td>10</td>
<td>+ dictionary</td>
<td>31.0</td>
<td>53.0</td>
<td>26.8</td>
</tr>
</tbody>
</table>

is a form of constrained decoding task. Note that we do not fully ensure that the constraint will apply. E.g., if the maximum attention never points to a source word that exist in the dictionary, the override mechanism is not triggered for that word.

Ideally, the dictionary should be generated by the user, and the quality of the final translation should also be measured by the user. To perform experiments that can be evaluated automatically without the need of user intervention, we create a simulated dictionary using the reference side of the development set. Since the reference is used, these experiments can be referred to as “cheating” experiments. The justification for using the reference is to be able to automatically measure the difference the dictionary is making using the Bleu and Ter automatic evaluation measures. If the dictionary is created without knowledge of the reference, the automatic metrics can penalize the system when it correctly uses a dictionary entry that mismatches the reference. We note that other works from the literature use the reference to automatically evaluate their experiments as well [Hasler & De Gisper\+ 18, Post & Vilar 18].

To create a dictionary, we map the reference to the source words using IBM4 alignment. We exclude English stop words, and only use source words aligned one-to-one to target words. We include up to 4 dictionary entries per sentence, and add reference translations only if they are not part of the baseline (i.e. unconstrained) translation, similar to [Hasler & De Gisper\+ 18]. We apply the same procedure to extract a dictionary for the baseline system and another for the alignment-based system. Each system has its own dictionary, while using comparable number of
We do this to extract reference entries that do not occur in the translation hypothesis of each system, which is done to make the use of these entries measurable through the BLEU and TER metrics. Table 5.18 shows the number of dictionary entries and the relative size with respect to the tokenized reference. Table 5.19 shows the results of using the dictionary in different systems on three language pairs. We note the following observations:

- The attention-based system (line 1) improves in BLEU when using the dictionary (line 2) in two cases, but this improvement is offset by a degradation in TER. One possible explanation for this is the following: the maximum attention weight at decoding step $i$ may point to a source word that is not immediately translated at step $i$. In other words, the maximum attention weight may create a wrong alignment, causing the override mechanism to replace the wrong target word. Regardless of its insertion position, the dictionary word can improve the unigram precision, which is used to compute BLEU. If the word is inserted at a wrong place in the translation, however, TER will penalize it, since shifting the word is required to match the reference. Note that enforcing a word, although it is the reference word, changes the context for the words to be generated after it, which may lead to searching regions not explored in the normal unconstrained search. The performance of the model in these worse-scoring regions may be worse than the original translation for the target words generated after the override. This may explain the longer translation we observe when enforcing words. In general, we observe longer translations are penalized more by TER than by BLEU.

- Dictionary override works better for the alignment-biased attention approach (lines 3, 4). We gain significant improvements on all tasks. Alignment-based search is different to standard neural machine translation search, as it hypothesizes all source positions at each decoding step, and it computes a different lexical model score for each of the source positions. This makes search more flexible to handle constraints. The higher performance compared to the pure RNN attention baseline can be due to this flexibility, which does not exist in the baseline, where at each decoding step a single vector of attention weights is computed, which determines the result of the next translation step. In the alignment-biased attention case, there are potentially $J$ possible attention weight vectors for a given decoding step. Another reason for the superior performance of the alignment-based systems is that they include explicit alignment information in training, and this can be beneficial in a task that is explicitly dependent on the alignment. We already observed that this system has a better alignment quality compared to the baseline.

- We also observe that the pure alignment-based RNN approach that does not include attention (lines 5, 6) exploits the dictionary to improve the performance. It is remarkable that the alignment-based system (line 6) outperforms the RNN attention baseline that includes the dictionary (line 2), although originally the RNN attention system (line 1) outperforms the RNN alignment systems (line 5). This holds for the English→Romanian and the Chinese→English systems. Note that the alignment error rate for the Chinese→English hard RNN alignment system is 23.5% compared to 52.3% for the RNN attention baseline (cf. Table 5.17).

- We observe improvements due to including the dictionary in the baseline transformer system on all language pairs. The multi-head transformer baseline is more capable of exploiting the dictionary than the single-head RNN attention baseline. This can be due to having multiple attention heads, in which case extracting the alignment is an accumulation over all heads; therefore, extracting alignment from multi-head attention can be more reliable compared to extracting it from a single attention component.
Table 5.20: A comparison between alignment methods used to determine the source word to be translated using the dictionary. The hard approach refers to using the hypothesized alignment during search while ignoring the attention weights. The soft approach uses both the attention weights and the hypothesized alignment to compute the alignment point, as given by Equations 5.11 and 5.12.

<table>
<thead>
<tr>
<th>#</th>
<th>system</th>
<th>WMT En→Ro newsdev2016</th>
<th>BOLT Zh→En DEV12-Tune</th>
<th>WMT De→En newstest2015</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BLEU [%]</td>
<td>TER [%]</td>
<td>BLEU [%]</td>
</tr>
<tr>
<td>1</td>
<td>RNN attention + alignment</td>
<td>25.6</td>
<td>56.5</td>
<td>22.1</td>
</tr>
<tr>
<td>2</td>
<td>+ dictionary (hard alignment)</td>
<td>27.6</td>
<td>55.5</td>
<td>23.5</td>
</tr>
<tr>
<td>3</td>
<td>+ dictionary (soft alignment)</td>
<td>27.9</td>
<td>55.4</td>
<td>24.4</td>
</tr>
<tr>
<td>4</td>
<td>transformer + alignment</td>
<td>27.4</td>
<td>55.2</td>
<td>24.3</td>
</tr>
<tr>
<td>5</td>
<td>+ dictionary (hard alignment)</td>
<td>29.3</td>
<td>54.0</td>
<td>26.8</td>
</tr>
<tr>
<td>6</td>
<td>+ dictionary (soft alignment)</td>
<td><strong>31.0</strong></td>
<td><strong>53.0</strong></td>
<td>26.3</td>
</tr>
</tbody>
</table>

- We achieve the best performance when using the dictionary with the alignment-assisted transformer on all three tasks. While it has comparable performance to the transformer baseline before introducing the dictionary (lines 7, 9), the alignment-assisted transformer exploits the dictionary to improve translation by a larger margin compared to the transformer baseline (lines 8, 10). Overall, we observe that the alignment-assisted transformer is better at inserting dictionary words compared to the transformer baseline. We argue that, given the comparable performance for regular translation, using the alignment-assisted transformer system for such alignment-based tasks outperforms the transformer baseline.

In Table 5.20, we compare between two alignment extraction methods for alignment-based systems. The hard approach refers to using the hypothesized source position during search as alignment point. The soft approach refers to using both the attention weights and the hypothesized position to compute the alignment point, as given by Equations 5.11 and 5.12. We compare the two approaches in constrained translation settings where the aligned source word should be translated according to the dictionary. The results are mixed for the RNN systems. Since these systems have a single attention component, the results are likely to vary depending on the quality of the alignment extracted from the attention component, and also depending on the fixed alignment used to train these models. When using multiple attention heads in the transformer case, the results vary less for German→English and Chinese→English, the two large tasks. The difference is still high for the English→Romanian system, which has a relatively small amount of training data. Based on these observations, we recommend referring to the alignment error rate before deciding between the soft and the hard approaches, as this decision is dependent on the task, the language pair, and the initial alignment quality used to train the system. We used the best approach in each case to report the results of Table 5.19.

### 5.7.13 Forced Alignment Training

The results presented so far are based on models trained using fixed IBM4 alignment. In this section, we investigate the effect of forced-alignment training, in which the training phase includes both alignment generation and training of lexical and alignment models. By training the neural models on alignment generated using neural networks, training is more consistent. We tackle the question whether having such consistent training has an impact on the translation performance. We follow the scheme described in Figure 5.2. We use the IBM4 alignment obtained through the tool GIZA++ as initial alignment to train the lexical and alignment models until convergence.
The models are then used to align the training data according to Equation 5.7. During forced-alignment, the target side is known; therefore, we leverage the benefits of the class-factored output layer in reducing calculation of the output layer softmax. This is done by limiting the evaluation of the output layer to the class layer, and to the word layer part that corresponds to the class of the known target word. The baseline models in the following experiments are the same ones presented in row 2 in Tables 5.10, 5.11, and 5.12.

In each step when a new alignment of the training data is obtained, we continue training the neural models using the previous set of parameters as initial weights, that is, we do not train the neural networks on the new alignment from scratch. We note that this is different from the conventional IBM1/HMM/IBM4 training, where the models are non-parametric count-based that are completely re-estimated using the newly generated alignment. We leverage the parameters of the neural network models in forced-alignment training by using them to initialize the models in each forced-alignment iteration. This reduces the training time of the neural networks during each iteration. We limit the neural network training to one (sub-)epoch to avoid overfitting the alignment of the current iteration. In the first split, the initial weights correspond to the complete training of the neural models using IBM4 alignment. Forced-alignment training can be performed on many levels. On the one extreme, it can be done on the batch-level, in which, new alignment is computed for every batch using the most recent neural network parameters. The other extreme is to re-align the whole training data at once. We choose a middle ground where we split the training data into 7 splits for the large corpora of WMT 2017 German→English and BOLT Chinese→English. We chose to do this to allow faster feedback between model training and alignment generation. In the case of the small WMT 2016 English→Romanian corpus, we align the whole training data at once. Another factor to consider when deciding on the split size is parallelization and hardware availability. Forced-alignment training is a heavily parallelizable task; therefore, the speed of aligning the training data in practice is mainly bound by hardware availability. We leverage 300 CPU nodes to align the training data in each iteration. While GPU decoding is faster per node, GPUs are more expensive and their availability is more limited than CPUs; therefore, using more CPU nodes is more appealing.

Forced-alignment training results for the WMT 2016 English→Romanian, BOLT Chinese→English, and WMT 2017 German→English tasks are presented in Tables 5.21, 5.22 and 5.23, respectively. We report the search score on the development set, which is a weighted log-linear combination of the lexical and alignment model probabilities. The baseline system uses training data aligned using IBM1/HMM/IBM4 training. This alignment is fixed for the baseline, but is changed during forced-alignment training to optimize the search criterion. During forced-alignment training, the objective is to align the training source and target sentences such that the search score is minimized, according to Equation 5.7. We observe that we have lower search cost for the system trained on the re-aligned data in comparison to the baseline system, indicating that forced-alignment training is able to generate alignment that has better search cost. In terms of translation quality, we observe slight improvements on the WMT 2016 English→Romanian and WMT 2017 German→English tasks, and larger improvements of up to 1.1% BLEU on the BOLT Chinese→English task.

To further analyze the re-aligned data, we gathered source fertility statistics of the training data, where we count how often a source word is aligned to a target word. We note that the overall number of alignment points is fixed, since we enforce that each target word is aligned to exactly one source word. Re-aligning the data during forced-alignment training can be viewed as finding a source word for each target word, which affects the source word fertility. The results are shown in Tables 5.24, 5.25, and 5.26 for WMT 2016 English→Romanian, BOLT Chinese→English, and WMT 2017 German→English, respectively. On both English→Romanian and German→English, we observe little change in the statistics when comparing the system trained on the IBM4 alignment and the one trained on the re-aligned data. There is, however, considerable change for the
5.7 Experimental Evaluation

Table 5.21: WMT 2016 English→Romanian forced-alignment training results.

<table>
<thead>
<tr>
<th>#</th>
<th>system</th>
<th>newdev2016</th>
<th></th>
<th>newtest2016</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>score</td>
<td>BLEU [%]</td>
<td>TER [%]</td>
</tr>
<tr>
<td>1</td>
<td>Fixed IBM4 alignment</td>
<td>1.89</td>
<td>23.7</td>
<td>59.6</td>
</tr>
<tr>
<td>2</td>
<td>+ re-alignment</td>
<td>1.51</td>
<td>23.9</td>
<td>59.5</td>
</tr>
</tbody>
</table>

Table 5.22: BOLT Chinese→English forced-alignment training results.

<table>
<thead>
<tr>
<th>#</th>
<th>system</th>
<th>DEV12-tune</th>
<th></th>
<th>DEV12-dev</th>
<th>P1R6-dev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>score</td>
<td>BLEU [%]</td>
<td>TER [%]</td>
<td>BLEU [%]</td>
</tr>
<tr>
<td>1</td>
<td>Fixed IBM4 alignment</td>
<td>1.89</td>
<td>16.7</td>
<td>69.1</td>
<td>16.0</td>
</tr>
<tr>
<td>2</td>
<td>+ re-alignment</td>
<td>1.48</td>
<td>17.6</td>
<td>69.3</td>
<td>16.7</td>
</tr>
</tbody>
</table>

Table 5.23: WMT 2017 German→English forced-alignment training results.

<table>
<thead>
<tr>
<th>#</th>
<th>system</th>
<th>newstest2015</th>
<th></th>
<th>newstest2017</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>score</td>
<td>BLEU [%]</td>
<td>TER [%]</td>
</tr>
<tr>
<td>1</td>
<td>Fixed IBM4 alignment</td>
<td>1.72</td>
<td>26.5</td>
<td>55.7</td>
</tr>
<tr>
<td>2</td>
<td>+ re-alignment</td>
<td>1.32</td>
<td>26.7</td>
<td>55.8</td>
</tr>
</tbody>
</table>

Table 5.24: Source coverage statistics of the training data for WMT 2016 English→Romanian.

<table>
<thead>
<tr>
<th>#</th>
<th>system</th>
<th>source fertility</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0 [%] 1 [%] 2 [%] 3 [%] &gt;=4 [%]</td>
</tr>
<tr>
<td>1</td>
<td>Fixed IBM4 alignment</td>
<td>17.6 65.8 14.4 1.8 0.4</td>
</tr>
<tr>
<td>2</td>
<td>+ forced-alignment</td>
<td>17.6 65.9 13.8 2.2 0.5</td>
</tr>
</tbody>
</table>

Table 5.25: Source coverage statistics of the training data for BOLT Chinese→English.

<table>
<thead>
<tr>
<th>#</th>
<th>system</th>
<th>source fertility</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0 [%] 1 [%] 2 [%] 3 [%] &gt;=4 [%]</td>
</tr>
<tr>
<td>1</td>
<td>Fixed IBM4 alignment</td>
<td>31.8 40.4 20.4 4.6 2.8</td>
</tr>
<tr>
<td>2</td>
<td>+ forced-alignment</td>
<td>36.1 33.9 18.7 7.4 3.9</td>
</tr>
</tbody>
</table>

Chinese→English system. Interestingly, this is the same system that achieves the largest improvement among the three tasks as shown in Table 5.22. We observe a significant increase in the number of unaligned source words, and a significant drop in the number of words aligned once. Overall, the system generates alignment where there are more unaligned source words. Since the number of alignment points is fixed, this means that the target words are aligned to a smaller number of source words.
Table 5.26: Source coverage statistics of the training data for WMT 2017 German→English.

<table>
<thead>
<tr>
<th>#</th>
<th>system</th>
<th>source fertility</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IBM4</td>
<td>0%</td>
</tr>
<tr>
<td>2</td>
<td>+ forced-alignment</td>
<td>21.3 59.9 15.7 2.3 0.8</td>
</tr>
</tbody>
</table>

5.7.14 Qualitative Analysis

Table 5.27 shows a selection of translation samples from the WMT 2017 German→English task for two systems: the transformer baseline, and the alignment-assisted transformer. The systems are close in performance in terms of BLEU and TER (cf. Table 5.8), with a slight advantage for the transformer baseline. The table includes the original German source sentence and the reference English translation. The sentences are tokenized and presented using words instead of subwords for readability, although the source is fed to neural network as subwords and the target is generated as subwords as well. The table shows positive examples where the alignment-assisted transformer system generates better translation than the baseline. In the 1st sample, the baseline system generates literal translation of the source, while the alignment-assisted system translates ‘nicht getan’ which literally means ‘not done’ to ‘not enough’, given the context. In the 2nd sample, the baseline generates the translation monotonically, using the same source word order. However, the alignment-assisted system reorders the sentence and generates a better translation. In the 3rd sample, the baseline generates ‘recruit’, which is a false translation of ‘abweisen’, which is split in the source sentence into ‘weisen’ and ‘ab’ in the middle and end of the sentence. The alignment-based system considers the two parts of the word to generate the correct translation ‘dismiss’, which is synonymous to the reference translation ‘turn away’. In the 4th sample, the last part of the source sentence ‘die sich lohnt’ is missing in the baseline translation, but is correctly translated to ‘that pays off’ by the alignment-assisted system. Note that the translation is also missing in the reference by mistake. The 5th sample shows that ‘Am imbiss’ is wrongly translated by the baseline to ‘the snack’ instead of ‘the snack bar’. The context of the German sentence is correctly used by the alignment-assisted system to disambiguate ‘Imbiss’ as a place, leading to the correct translation ‘the snack bar’.

We also selected negative examples where the alignment-assisted system generates wrong or inaccurate translations. The sample translations are shown in Table 5.28. In the 1st sample, the verb ‘indicted’ is attributed to ‘Donald Trump’ as a subject, while the original source sentence is given in a passive form with no subject. Note that the baseline also generated a wrong translation in this case. Note the generation of the unknown symbol ‘<unk>’, which is included in the training data, since the vocabulary is limited to the most frequent 50k subword units. The rest of the subwords are mapped to ‘<unk>’. The unknown symbol in the target is replaced by the aligned unknown source word, if any. In this example, there is no unknown source word; therefore, the generation of the ‘<unk>’ symbol is not warranted. For the 2nd sentence, a monotone English translation of the first half of the sentence is: ‘praised is the willingness of those who are responsible for construction work, ...’. The alignment-based system translates ‘Entgegenkommen’ which means ‘willingness’ to ‘to be willing’, which is left without a preposition, breaking the construction of the sentence. The baseline also incorrectly translates the first part of the sentence. In the 3rd sentence, the name ‘Glee’ of a TV show is incorrectly translated to ‘track’ by the alignment-assisted system. The baseline translates this correctly. In the 4th sample, there is a typo in the source sentence in the word ‘aun’, which should be ‘aus’, meaning ‘of’. The alignment-assisted system gets the translation of this word wrong, generating the translation ‘...
Table 5.27: Sample translations from the WMT German→English transformer baseline system and the transformer-based alignment-assisted system (Table 5.8, rows 5 and 6). The table also includes the source German sentence being translated, along with the English reference. The table shows positive cases where the alignment-assisted transformer provides better translations compared to the transformer baseline. Negative examples are provided in Table 5.28.

<table>
<thead>
<tr>
<th></th>
<th>source</th>
<th>reference</th>
<th>transformer</th>
<th>+ alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Doch nur mit Wohnungen ist es nicht getan .</td>
<td>But homes alone will not do the trick .</td>
<td>But it is only with housing that it is not done .</td>
<td>But only housing is not enough .</td>
</tr>
<tr>
<td>2</td>
<td>Große Stütze für VW ist der chinesische Markt .</td>
<td>The Chinese market is a source of great support for VW .</td>
<td>A large support for VW is the Chinese market .</td>
<td>The Chinese market is a big support for VW .</td>
</tr>
<tr>
<td>3</td>
<td>Gemeinderechtszentren weisen jährlich 160.000 Leute ab</td>
<td>Community legal centres turn away 160,000 people a year</td>
<td>Councillors recruit 160,000 people each year</td>
<td>Community centres dismiss 160,000 people a year</td>
</tr>
<tr>
<td>4</td>
<td>Pokémon Go , eine Jagd nach Gesundheit und Glück , die sich lohnt</td>
<td>Pokémon Go , a treasure hunt for health and happiness</td>
<td>Pokémon Go , a treasure hunt for health and happiness</td>
<td>Pokémon Go , a hunt for health and happiness that pays off</td>
</tr>
<tr>
<td>5</td>
<td>Am Imbiss und am angrenzenden Gebäude entstand ein Schaden von 10000 Euro .</td>
<td>Damage amounting to 10,000 euros was caused to the snack bar and the neighbouring building .</td>
<td>On the snack and on the adjacent building a damage of EUR 10000 was caused .</td>
<td>A damage of EUR 10000 was caused to the snack bar and adjacent building .</td>
</tr>
</tbody>
</table>

provide communities with countless, young girls ...’ instead of ‘provide communities of countless, young girls ...’. The baseline however overcomes the typo and translates the sentence correctly.

5.8 Contributions

The author proposed the alignment-based decoding framework in [Alkhouli & Bretschner+ 16], where feedforward lexicon and feedforward alignment models were initially used. The search algorithm is an extension of the simple beam search algorithm used in [Bahdanau & Cho+ 14, Sutskever & Vinyals+ 14, Cho & van Merrienboer+ 14a]. The author contributed the idea and the
Table 5.28: Sample translations from the WMT German→English transformer baseline system and the transformer-based alignment-assisted system (Table 5.8, rows 5 and 6). The table also includes the source German sentence being translated, along with the English reference. The table shows negative cases where the alignment-assisted transformer provides worse translations compared to the transformer baseline. Positive examples are provided in Table 5.27.

<table>
<thead>
<tr>
<th></th>
<th>source</th>
<th>reference</th>
<th>transformer</th>
<th>+ alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Donald Trump Wahlkampagne angeklagt nachdem Mitarbeiter angeblich Waffe zog</td>
<td>Donald Trump campaign sued after staffer allegedly pulled a gun</td>
<td>Donald Trump’s Election Campaign &lt;unk&gt; of Staff Recalling Weapons</td>
<td>Donald Trump indicted election campaign after staff allegedly pulled a gun</td>
</tr>
<tr>
<td>2</td>
<td>Gelobt wird auch das Entgegenkommen der für die Bauarbeiten Verantwortlichen, etwa wenn es um die Einrichtung von Behelfseinfahrten geht.</td>
<td>The obligingness of the people responsible for the construction works in terms of setting up substitute driveways.</td>
<td>It also provides for the accommodation of those responsible for the construction works, for example when it comes to the establishment of temporary driveways.</td>
<td>There is also a need for those responsible for the construction work to be willing, for example when it comes to the installation of container driveways.</td>
</tr>
<tr>
<td>3</td>
<td>Und im Schwimmbekken wird der südafrikanische Schwimmer Cameron van der Burgh seit Jahren mit dem Glee Star Matthew Morrison verglichen.</td>
<td>And over in the pool, South African swimmer Cameron van der Burgh has been getting compared to Glee star Matthew Morrison for years now.</td>
<td>And in the swimming pool, South African swimmer Cameron van der Burgh has been compared to Glee Star Matthew Morrison for years.</td>
<td>And in the swimming pool, South African swimmer Cameron van der Burgh has been compared for years with the track Star Matthew Morrison.</td>
</tr>
<tr>
<td>4</td>
<td>Wir müssen den Gemeinden aun unzähligen, jungen Mädchen, die von Simone, Gabby und Laurie inspiriert werden, ein leistbareres und leichter zugängliches Turntraining ermöglichen.</td>
<td>We must bring more affordable and accessible gymnastics training to the communities of countless young girls who are inspired by Simone, Gabby and Laurie.</td>
<td>We must provide communities with more affordable and easily accessible training for young girls inspired by Simone, Gabby and Laurie.</td>
<td>We need to provide communities with countless, young girls inspired by Simone, Gabby and Laurie with a more affordable and easily accessible exercise.</td>
</tr>
</tbody>
</table>

design of the search algorithm. The decoder was implemented by the author in Sockeye. Gabriel Bretschner implemented it in Jane as part of his Bachelor thesis which was supervised by the author. The model combinations used in the alignment-based framework described in Section 5.6.1 are contributions of the author. The forced-alignment training algorithm is proposed by the author. The author implemented the initial version of forced alignment into Jane, which was later extended by Gabriel Bretschner during his bachelor thesis and Mohammed Hethnawi during his master thesis, both were supervised by the author. The author implemented a forced-alignment framework that consists of the full pipeline described in Figure 5.2. All forced-alignment training experiments reported in this work use the Jane toolkit.
The author proposed the search pruning techniques of Section 5.6.4 in [Alkhouli & Bretschner+ 16, Alkhouli & Bretschner+ 18]. Gabriel Bretschner implemented class-based lexical pruning into \textit{rwthlm} and alignment pruning into \textit{Sockeye} during his bachelor and master theses. The soft and hard alignment extraction methods described in Section 5.6.5 are contributions of the author [Alkhouli & Ney 17, Alkhouli & Bretschner+ 18].
6. Phrase-Based Machine Translation

The phrase-based approach was the state-of-the-art machine translation approach until 2014. In this dissertation, we discuss the integration of neural networks into phrase-based systems. The integration we will focus on applies neural networks as additional features within the phrase-based system. That is, neural networks are complimentary components: they are not used to generate translation directly, but rather used to score translation hypotheses that are generated by the underlying phrase-based system. This is in contrast to neural machine translation where the neural network is used to generate the translation hypotheses. We will first introduce the phrase-based machine translation paradigm in Section 6.1 and discuss the main models the system is composed of in Section 6.3. We will give an overview of the search procedure used to generate phrase-based translation hypotheses in Section 6.4. Afterwards, we will present how to integrate some of the neural network models described in Chapter 4 into the phrase-based pipeline. We discuss two ways of integration: first-pass and second-pass. In the first-pass integration, the neural networks are used to score the partial hypotheses online during the translation process. This is covered in Section 6.5. In the second-pass integration, the neural networks are applied offline to score and re-rank complete translation hypotheses, as discussed in Section 6.6. We present our results in Section 6.7.

6.1 Introduction

Early statistical machine translation systems relied on the IBM models proposed in [Brown & Della Pietra+ 93] and the hidden Markov model [Vogel & Ney+ 96] approach. These models vary in terms of the context they are conditioned on. While simple models are conditioned only on the word being translated, one can include more complex concepts such as fertility, which models the number of words in one language that are generated by a word from the other language. All these models are word-based — they generate one word at each step. Later on, the alignment template approach [Och & Tillmann+ 99, Och & Ney 04] was proposed, which inspired the phrase-based translation paradigm [Zens & Och+ 02, Koehn & Och+ 03]. These systems became widely adopted as the state-of-the-art machine translation systems for over a decade, until neural machine translation was introduced.

Phrase-based models differ from word-based models in that they score a whole phrase at each step. Take the phrase “it is raining cats and dogs”, for instance. To generate such a phrase, word-based models score each word separately, and the scores are combined to give a score for the phrase. In contrast, phrase-based models have the possibility to score the whole phrase at once. For word-based models to generate such a phrase, they have to model long context, and search should be flexible enough not to terminate partial hypotheses leading to such translation. Phrase-based systems, on the other hand, need only to store such an entry in a phrase table. During search, the whole phrase can be hypothesized as one atomic unit. For these reasons, the
phrase-based system is more appealing to model phrases and language-specific idioms. Figure 6.1 and Figure 6.2 respectively show word-based and phrase-based translation examples.

### 6.2 State of The Art

This chapter focuses on using recurrent neural networks to score phrase-based translation candidates, either online during decoding, or in \( N \)-best rescoring. On the topic of integrating recurrent models into phrase-based decoding, [Auli & Galley + 13] propose to store the recurrent hidden states in the search states, and to ignore them when comparing search states to decide on equivalency for state recombination. State recombination will not be affected, however, computing the recurrent model scores using this approach becomes approximate, since only the hidden state corresponding to the best path is kept when recombining the nodes. This means that the target hypotheses that differ in the early history are treated as if they have identical hidden states, assuming hidden states are mostly affected by the recent words used to compute them. The authors apply the method in second-pass lattice rescoring setups. In comparison, the integration experiments presented in this chapter include model scoring in first-pass decoding, as proposed in [Alkhouli & Rietig + 15]. [Schwenk 12] uses feedforward models to compute additional phrase scores. Although the neural phrase scores are used in first-pass decoding, modeling is limited to the phrase boundary, leaving the remaining sentence context surrounding the phrase unused. [Vaswani & Zhao + 13] use noise contrastive estimation to score feedforward language models in first-pass decoding. The method avoids computing the costly output normalization step over the target vocabulary. [Auli & Gao 14] use an expected BLEU criterion instead of cross-entropy. They train recurrent neural language models without the need to normalize the output layer, but training becomes computationally more expensive as each training example is an \( N \)-best list instead of a sentence. At decoding time, however, scoring with the neural network is faster since normalization is not needed. Furthermore, they integrate RNNs trained using cross-entropy without affecting state recombination. They report results over a baseline having a language model trained on the target side of the parallel data. [Devlin & Zhib + 14] augment the cross-entropy training objective function to produce approximately normalized scores directly. They also precompute the first hidden layer beforehand, resulting in large speedups. In this thesis, we use RNNs trained using cross-entropy with a class-factored output layer to reduce the normalization cost. Since the target word is known during rescoring, this reduces the normalization cost drastically.
During first-pass decoding, the normalization cost is also reduced, since the models are used to score phrase candidates, that is, the target phrase is known, which results in a setup similar to rescoring. However, due to the large number of hypothesized phrase candidates during decoding, scoring using the neural model is still expensive compared to looking up offline phrase scores. This holds true regardless of the method used to tackle the costly output normalization problem.

N-best rescoring is a second-pass scoring step applied after generating full translation candidates in a first-pass decoding step. [Le & Allauzen+ 12] present translation models using an output layer with classes and a shortlist for rescoring using feedforward networks. [Kalchbrenner & Blunsom 13] use recurrent neural networks with full source sentence representations obtained by applying a sequence of convolutions on the source sentence. The source representation feeds into a hidden layer that is recurrent over the target words. The model applied in rescoring resulted in no improvements compared to the state-of-the-art at the time. [Auli & Galley+ 13] report results using recurrent language models applied in N-best and lattice rescoring setups. Phrase-level models are introduced in [Hu & Auli+ 14] and [Sundermeyer & Alkhouli+ 14] to rescore N-best lists.

6.3 Model Definition

A source sentence \( f^l \) can be segmented into \( K \) source phrases \( f^K = f_1 \ldots f_k \ldots f_K \). Similarly, if \( e^l \) is the translation corresponding to the source sentence, it can be segmented into \( K \) target phrases \( e^K = e_1 \ldots e_k \ldots e_K \). We refer to the pair \( (f_k, e_k) \) as a phrase pair. Let \( s^K \) denote the segmentation of the sentence pair into \( K \) phrase pairs, where the \( k \)-th segment is a triplet \( s_k = (i_k; b_k, j_k) \), where \( i_k \) is the ending position of the \( k \)-th target phrase \( e_k \), and \( b_k \) and \( j_k \) are the starting and ending positions of the source phrase \( f_k \), respectively. The target phrases are assumed to have monotone order. As a result, \( f_k = f_{b_k} \ldots f_{j_k} = f_{b_k}^j \) and \( e_k = e_{i_k+1} \ldots e_{i_k} = e_{i_k+1}^j \). Figure 6.3 shows an example of phrase segmentation of the phrasal translation of Figure 6.2.

There are constraints that need to be satisfied for a segmentation to be valid. All source words and all target words should be covered by exactly one phrase. Formally,

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

\[
\{b_k, \ldots, j_k\} \cap \{b_{k'}, \ldots, j_{k'}\} = \emptyset \quad \forall \ k \neq k'.
\]

In addition, the following constraints need to be satisfied:

\[
i_0 = j_0 = b_0 = 0
\]

\[
i_{K+1} = I + 1
\]

\[
b_{K+1} = j_{K+1} = J + 1
\]

\[
e_0 = <s>
\]

\[
e_{I+1} = </s>.
\]

where we introduce an additional segmentation \( s_0 \) covering the sentence begin symbol \(<s>\) and another segmentation \( s_{K+1} \) covering the sentence end symbol \(</s>\).
Figure 6.3: Illustration of the phrase segmentation corresponding to the translation shown in Figure 6.2. The shaded areas represent the word alignment. The example shows four phrase pairs used to generate the translation, with their corresponding boundaries.

Formally, the phrase segmentation $s^K$ is introduced as a hidden variable, and the posterior probability is computed as a marginalization over all possible segmentations of different lengths $K$

$$
Pr(e^I_1, s^K | f^J_1) = \sum_{K,s^K} Pr(e^I_1, s^K | f^J_1)
$$

$$
= \sum_{K,s^K} \frac{\exp \left( \sum_{m=1}^M \lambda_m h_m(e^I_1, s^K, f^J_1) \right)}{\sum_{\tilde{K},s^{\tilde{K}}} \exp \left( \sum_{m=1}^M \lambda_m h_m(\tilde{e}^I_1, s^{\tilde{K}}, f^J_1) \right)}
$$

Here, the joint probability distribution $Pr(e^I_1, s^K | f^J_1)$ is modeled by a log-linear model composed of $M$ models $h_m(.,.,.)$. Since the denominator is independent of $e^I_1$, it is not required to compute the maximizing target string and can be dropped; therefore, Bayes’ decision rule using this model combination can be formulated as follows:

$$
f^J_1 \rightarrow \tilde{e}^I_1(f^J_1) = \arg\max_{I, e^I_1} \left\{ Pr(e^I_1 | f^J_1) \right\}
$$

$$
= \arg\max_{I, e^I_1} \left\{ \sum_{K,s^K} \exp \left( \sum_{m=1}^M \lambda_m h_m(e^I_1, s^K, f^J_1) \right) \right\}.
$$

In practice, the sum over all segmentations is replaced by the maximum. The decision rule with the maximum approximation is

$$
f^J_1 \rightarrow \tilde{e}^I_1(f^J_1) = \arg\max_{I, e^I_1} \left\{ \max_{K,s^K} \sum_{m=1}^M \lambda_m h_m(e^I_1, s^K, f^J_1) \right\}, \quad (6.1)
$$
6.3 Model Definition

where the exponential function is dropped because it is strictly monotone. The best translation is generated by searching over all possible translations and their underlying phrase segmentation. In practice, the search for the maximum is not exact, and an approximate search procedure called beam search is used to find it.

We will now discuss the different models $h_m(e_1^s, s_k^I, f_j^f)$ that are the main components of a phrase-based system. These models can be classified into three main categories: (1) translation modeling that includes both source and target information, (2) language modeling that only includes target language information, and (3) alignment modeling that models inter-phrase reordering, which is also referred to as phrase alignment.

6.3.1 Phrase Translation Models

Phrase translation models compute phrase-level scores: given a phrase pair $(\tilde{f}, \tilde{e})$, the model computes scores for the given phrase pair. These scores are computed as relative frequencies using counts. Let $N(\tilde{f}, \tilde{e})$ denote the count of the phrase pair $(\tilde{f}, \tilde{e})$ in the bilingual training corpus, and $N(\tilde{e}) = \sum_{\tilde{f}} N(\tilde{f}, \tilde{e})$ denote the count of the phrase $\tilde{e}$ in the target corpus, then

$$p(\tilde{f}|\tilde{e}) = \frac{N(\tilde{f}, \tilde{e})}{N(\tilde{e})}$$

defines the target-to-source phrase model. Analogously, the source-to-target model is defined as follows:

$$p(\tilde{e}|\tilde{f}) = \frac{N(\tilde{f}, \tilde{e})}{N(\tilde{f})},$$

where $N(\tilde{f}) = \sum_{\tilde{e}} N(\tilde{f}, \tilde{e})$ denotes the count of the phrase $\tilde{f}$ in the source corpus. Using these models, the source-to-target (s2t) and target-to-source (t2s) phrase scores of the sentence pair is given by

$$h_{s2t}(e_1^s, s_k^I, f_j^f) = \log \prod_{k=1}^{K} p(\tilde{e}_k|\tilde{f}_k) = \sum_{k=1}^{K} \log p(\tilde{e}_k|\tilde{f}_k)$$

$$h_{t2s}(e_1^s, s_k^I, f_j^f) = \log \prod_{k=1}^{K} p(\tilde{f}_k|\tilde{e}_k) = \sum_{k=1}^{K} \log p(\tilde{f}_k|\tilde{e}_k).$$

Note that the phrase score is only dependent on the phrase pair and is independent of its surrounding context. The phrase scores are typically stored in an offline table and looked up and combined to obtain the sentence-level score.

6.3.2 Word Lexicon Models

While the phrase translation models operate at the phrase level, word lexicon models operate within phrases at the word level. They are based on source-to-target and target-to-source word probability models $p(e|f)$ and $p(f|e)$, which are the IBM 1 models in both directions introduced in Section 3.2.2. The models are defined as follows:

$$h_{w2t}(e_1^s, s_k^I, f_j^f) = \sum_{k=1}^{K} \sum_{i=ik_{k-1}+1}^{ik_k} \log \left( p(e_i|f_0) + \sum_{j=b_k}^{j_k} p(e_i|f_j) \right)$$

$$h_{wt2s}(e_1^s, s_k^I, f_j^f) = \sum_{k=1}^{K} \sum_{j=b_k}^{jk_k} \log \left( p(f_j|e_0) + \sum_{i=ik_{k-1}+1}^{ik_k} p(f_j|e_i) \right).$$
where \( f_0 \) and \( e_0 \) denote the source and target empty words, which imply that the word is not aligned to an actual word on the opposite side.

### 6.3.3 Phrase-Count Indicator Models

Phrase-count indicator models are binary features that reflect how often a phrase pair is seen in training. Given a positive integer threshold \( \tau \in \mathbb{N}^+ \), the sentence score using the indicator model is given by

\[
h_{C,\tau}(e^I_1, s^K_1, f^J_1) = \sum_{k=1}^K h_{\tau}(\tilde{f}_k, \tilde{e}_k),
\]

where the score of each phrase is defined as follows:

\[
h_{\tau}(\tilde{f}_k, \tilde{e}_k) = \begin{cases} 
0, & \text{if } N(\tilde{f}_k, \tilde{e}_k) \leq \tau, \\
1, & \text{otherwise.} 
\end{cases} \tag{6.2}
\]

\( \tau \) is typically set to 1, 2, and 3.

### 6.3.4 Enhanced Low-Frequency Model

Since phrase translation models are based on relative frequencies, they do not account for how often a phrase pair occurs in the training data. To penalize rare phrase pair events, the enhanced low-frequency (elf) model is used [Chen & Kuhn +11]. It is inversely proportional to the phrase pair count

\[
h_{\text{elf}}(e^I_1, s^K_1, f^J_1) = \sum_{k=1}^K \frac{1}{N(\tilde{f}_k, \tilde{e}_k)}.
\]

### 6.3.5 Length Models

We use a simple length model counting the number of target words used to control the length of the translation hypothesis \( I \). The model is referred to as word penalty (wp)

\[
h_{\text{wp}}(e^I_1, s^K_1, f^J_1) = \sum_{k=1}^K |\tilde{e}_k| = I,
\]

where \( |\tilde{e}_k| \) is the number of words in the phrase \( \tilde{e}_k \). Another penalty is the phrase penalty which is used to control the number of hypothesized phrases \( K \), which implicitly controls the phrase length

\[
h_{\text{pp}}(e^I_1, s^K_1, f^J_1) = \sum_{k=1}^K 1 = K.
\]

### 6.3.6 Count-Based Language Models

The count-based \( n \)-gram language model (LM) is an essential component in phrase-based systems, and it is used to capture the dependencies in the target language. The \( n \)-gram language model indicated by \( p(e_i | e^{i-n+1}_{i-n+1}) \) predicts the probability of the current target word conditioned on the \( n - 1 \) previous target words as context. The context can span multiple phrases, and therefore it is important to keep track of the full context needed to compute the language model scores. The
language model is trained on the monolingual target corpus, which is typically much larger than
the bilingual corpus. Training language models using the maximum likelihood criterion leads to
relative frequency estimates that assign zero probability to unseen events. We use language mod-
els with interpolated modified Kneser-Ney discounting [Kneser & Ney 95, Chen & Goodman 98]
to smooth the probability distribution such that it can account for events unseen during training.
The sentence probability is given by

\[ h_{LM}(e_I^1, s^K_1, f_J^1) = \sum_{i=1}^{I+1} \log p(e_i|e_{i-n+1}^{i-1}). \]  

(6.3)

We used the SRILM [Stolcke 02] and the KenLM toolkits [Heafield 11] to train the language
models.

### 6.3.7 Reordering Models

The distortion model is a global reordering model that influences phrase reordering. We use a
distortion model that penalizes deviations from monotone alignment linearly, up to a jump width
limit \( D \), after which the penalty is quadratic. The distance between the \((k - 1)\)-th and the \( k \)-th
phrase is defined as follows:

\[ q_{dm}(b_k, j_{k-1}) = \begin{cases} |b_k - j_{k-1} - 1| & \text{if } |b_k - j_{k-1} - 1| < D, \\ |b_k - j_{k-1} - 1|^2 & \text{otherwise}. \end{cases} \]  

(6.4)

According to this function, the monotone jump where \( b_k = j_{k-1} + 1 \) has the smallest distance.
The sentence-level score is

\[ h_{dm}(e_I^1, s^K_1, f_J^1) = \sum_{k=1}^{K+1} q_{dm}(b_k, j_{k-1}), \]

which covers jumping to the sentence end symbol, the \((K + 1)\)-th segment.

We also use the hierarchical reordering model [Galley & Manning 08, Cherry & Moore 12],
which distinguishes between three classes of phrase reordering: monotone, swap, and discontinu-
ous. The orientation of the phrase is determined with respect to the largest block that contains
the previous phrase, where a block contains phrase pairs such that all phrases are entirely inside
or outside the block. We use models that compute the orientation from left to right and from
right to left. Together with the three orientation classes, this yields six hierarchical reordering
models in total. The models are estimated as relative frequencies.

In the next section, we will provide the search procedure including the phrase translation
models, the word lexicon models, the distortion model, the \( n \)-gram language model and the
length models described in this section.

### 6.4 Search

Now that the models are defined, we will discuss the problem of search, which is also called
decoding. The search problem is to find the best translation candidate according to Equation 6.1.
Search is implied by the max and argmax operators that search for the best translation among the
different translations \( e_I^1 \) of unknown target length \( I \), that have the underlying segmentations \( s^K_1 \)
which have \( K \) segments. A naive enumeration of all candidates has an exponential time complexity
in the source sentence length. Since the models introduced in the previous section have relatively
short context dependency, this allows to structure search using dynamic programming [Bellman
6 Phrase-Based Machine Translation

Figure 6.4: Excerpt of a search graph for the source sentence from Figure 6.2 [Rietig 13]. The first entry in each node contains the coverage set \( C \) represented as a bit vector. If the \( j \)-th bit is 1, then the \( j \)-th source word is covered. The second entry contains the language model history and the last translated source position. The third entry contains the partial hypothesis score. The arrows are labeled with the target phrase, and the indices of the source phrase.

57]. Dynamic programming decomposes the search problem into incremental steps, where each step builds on the results of the previous step. This can reduce the number of paths explored in search without introducing any approximation. Still, the search problem is NP-hard [Knight 99]. To alleviate this, we apply beam search [Jelinek 97] to prune the search candidates and focus the effort on the most viable ones. This is an approximation and does not guarantee finding the optimal translation according to the models. This approximation leads to a search error. In this dissertation, we use the source cardinality synchronous search [Zens & Ney 08], which we will describe next.

6.4.1 Search Graph

Search can be represented by a graph of nodes and edges. A search node represents a partial translation, and an edge represented by an arch extending from a previous node to a new node. The edge is labeled with the target phrase that extends the partial hypothesis of the previous node to the new partial hypothesis of the new node. The edge is labeled with \((\tilde{e}_k, b_k, j_k)\), which includes the indices of the source phrase \( f_{b_k} ... f_{j_k} \) being translated to \( \tilde{e}_k \). The final translation is represented by the best scoring path that extends from the initial node to the final node that translates the sentence end symbol. A snippet of a search graph is shown in Figure 6.4.

To avoid overlaps, each node keeps track of the source words that were covered by the path leading to it. This is referred to as the coverage set \( C \subseteq \{1, ..., J\} \). The initial node has an empty coverage set \( C = \{\} \), and a final node has all source words covered, i.e. \( C = \{1 ... J\} \). Each node should maintain a search state, including all the information needed by the models to extend it to a new partial translation. E.g., an \( n \)-gram language model is conditioned on the \( n - 1 \) previous target words; therefore, each node should include the \( n - 1 \) target history \( e' \) to allow computing the language model scores. Computing the distortion penalty requires knowing the last translation source position \( j \); therefore, each node stores the triplet \((C, e', j)\). Note that the phrase and word translation models described in Section 6.3 are local, i.e. they do not depend on previous context, and therefore no extra information needs to be stored in the nodes to compute them.
6.4 Search

The search procedure proceeds by hypothesizing target phrases monotonously from left to right. At each step, a source span \( \{b...j\} \) is selected with the constraint that it has no overlap with the previous coverage set, i.e. \( C' \cap \{b...j\} = \{\} \), and a target phrase \( \tilde{c} \) is hypothesized as its translation. The search state is \( C = C' \cup \{b...j\} \). Let \( \tilde{c} \oplus \tilde{e} \) denote the new language model history. The new search state is \( (C' \cup \{b...j\}, \tilde{c} \oplus \tilde{e}, j) \). During search, several nodes can end up with the same search state. Since the state represents all the information needed to compute future expansions, recombination is used to merge all equivalent nodes. This is illustrated in Figure 6.4 in the case of two edges pointing to the node that has “The cat” as target history. The score of the best partial recombined hypothesis is kept as the score of the recombined node.

We will now introduce some auxiliary quantities needed to compute the score of a newly created search node. Let \( q_{TM}(\tilde{c}, b, j) \) be the local translation score combining phrase and word lexicon model scores, defined as follows:

\[
q_{TM}(\tilde{c}, b, j) = \lambda_{s2t} \log p(\tilde{c} | f_b ... f_j) + \lambda_{t2s} \log p(f_b ... f_j | \tilde{c}) + \sum_{j' = b}^{j} \log p(f_{j'} | e_0) + \sum_{i=0}^{k} p(f_{j'} | \tilde{e}_i) + \lambda_{w2t} \log p(\tilde{e}_i | f_0) + \sum_{j' = b}^{j} p(\tilde{e}_i | f_{j'})
\]

(6.5)

where \( \lambda_{s2t} \) and \( \lambda_{t2s} \) are the log-linear weights respectively associated with the source-to-target and target-to-source phrase translation models. \( \lambda_{w2t} \) are the weights respectively associated with the source-to-target and target-to-source word lexicon models. In addition, we need to compute language model scores using the auxiliary quantity \( q_{LM}(\tilde{c} | \tilde{e}') \) defined using an n-gram language model as follows:

\[
q_{LM}(\tilde{c} | \tilde{e}') = \lambda_{LM} \cdot \sum_{i=1}^{n} \log p(\tilde{e}_i | \tilde{h}_i),
\]

(6.6)

where \( \tilde{h}_i = \tilde{e}_{i-1} ... \tilde{e}_1 \tilde{e}' \) is the language model context. If \( |\tilde{h}_i| > n - 1 \), then the effective context is limited to the most recent \( n - 1 \) target words.

When a new target phrase is hypothesized as a translation of a source phrase starting at source position \( b \), the distortion model score is \( q_{DM}(b, j') \), computed using the last translated source position \( j' \) (c.f. Equation 6.4). The score including the log-linear weight is given by

\[
q_{DM}(b, j') = \lambda_{DM} \cdot q_{dm}(b, j').
\]

(6.7)

The score of a new node is dependent on the score of the path that leads to \( Q(C', \tilde{e}', j') \) and on its local score. The score is computed using the following dynamic programming recursive equations:

\[
Q(\{\}, <s>, 0) = 0
\]

\[
Q(C, \tilde{c}, j) = \max_{\substack{(C', \tilde{e}', j'), (\tilde{e}', b, j) : \ C' \cap \{b...j\} = \{\} \\wedge \tilde{c} \oplus \tilde{e}' \in E(\tilde{c}, b, j) \}} \left\{ Q(C', \tilde{e}', j') + q_{TM}(\tilde{e}', b, j) + q_{LM}(\tilde{e}' | \tilde{e}') + q_{DM}(b, j') \right\},
\]

(6.8)

where \( E(b, j) \) is the set of phrase translation candidates for the source phrase \( \tilde{f} = f_b ... f_j \). The maximum is performed over all possible splits of the target string \( \tilde{c} \), with the constraint that a target phrase from the phrase table is used, and that no translation overlap occurs.
The final translation score includes the distortion model score for jumping to the sentence end symbol, and the language model score

\[
\hat{Q} = \max_{\tilde{e}, \tilde{j}} \left\{ Q(\{1, \ldots, J\}, \tilde{e}, \tilde{j}) + q_{LM}(\langle \rangle | \tilde{s}) + q_{DM}(J + 1, j) \right\}.
\]  
(6.12)

6.4.2 Search Pruning

Even after structuring translation as a dynamic programming problem, search still has an exponential complexity: if the source sentence is of length \(J\), there are \(2^J\) distinct coverage sets. To reduce the exponential complexity, _beam search_ is applied. Beam search prunes the search space focusing the search effort on a small part of the search graph. Histogram pruning is a common pruning approach which keeps track of a fixed number of candidates during search [Steinbiss & Tran+ 94]. This can be applied on several levels. Below we describe pruning on the coverage set level, the coverage cardinality level, and the phrase level.

** Lexical pruning per coverage:** For each coverage set \(C\), keep track of the top \(N_l\) candidates. These candidates may be different in the last translated source position and the language model history. For each coverage vector \(C\), the top scoring candidate is given by

\[
Q(C) = \max_{\tilde{e}, \tilde{j}} \left\{ Q(C, \tilde{e}, \tilde{j}) + R(C, \tilde{j}) \right\}.
\]

The term \(R(C, \tilde{j})\) is the rest cost estimate, which estimates the future cost of the hypothesis, used to avoid translating the easy parts of the source sentence first. For more details on rest cost estimation, we refer the reader to [Zens 08], pp. 57–59.

** Reordering pruning per coverage cardinality:** For each coverage cardinality \(|C|\), keep track of the top \(N_r\) coverage candidates. This allows comparing lexical hypotheses with the same number of translated source words that have different translation order. Since each coverage set has its associated lexical hypotheses, if a coverage set is pruned, all of its hypotheses are pruned along with it. The score of a coverage set cardinality \(c\) is computed as follows:

\[
Q(c) = \max_{C: |C| = c, \tilde{e}, \tilde{j}} \left\{ Q(C, \tilde{e}, \tilde{j}) + R(C, j) \right\}.
\]

** Observation pruning:** This limits the number of phrase translation hypotheses for any source phrase to \(N_o\) phrases. Phrase scoring includes language model scores that disregard context beyond the phrase itself

\[
Q(\hat{f}, \tilde{e}) = q_{TM}(\tilde{e}, b, j) + q_{LM}(\tilde{e} | \langle \rangle).
\]

Observation pruning is done offline and the phrases are stored in sorted order according to their scores.

** IBM Reordering Constraint:** The IBM constraint for word-based systems [Berger & Brown+ 96] is applied to the phrase-based system. Search is disallowed from hypothesizing a phrase if it creates more than \(N_g\) contiguous source gaps (uncovered positions). The constraint is given by

\[
|\{j > 1 | j \in C \land j - 1 \notin C\}| \leq N_g.
\]

6.5 Neural Network Integration

In this section, we will discuss integrating neural models into the phrase-based search algorithm. Evaluating neural networks is computationally demanding; therefore, naive integration of neural
networks into phrase-based search can be inefficient. We discuss the integration of the recurrent neural network language model (c.f. Section 4.5.4), the recurrent neural network lexical model (c.f. Section 4.5.5), and the bidirectional recurrent neural network lexical model without target information (c.f. Section 4.5.6).

### 6.5.1 Recurrent Neural Network Language Model

Unlike the \( n \)-gram language model that is conditioned on fixed target history, recurrent language models are conditioned on the full target history \( e_{i-1} \); therefore, computing recurrent language models during search requires the search states to include the complete target history. As discussed in Section 6.4.1, equivalent search states are recombined during search, and equivalency is determined based on the triplet \( (C, \hat{e}, j) \), where \( \hat{e} \) is the target history needed to evaluate the language model. If \( \hat{e} = e_{i-1} \) represents the full target hypothesis, there will be fewer equivalent states during search, which means less recombination. The purpose of recombination is to compress the beam entries, making space for other entries that correspond to new regions in the search space. Less recombination therefore leads to less regions of the search space being explored under the same pruning conditions. This leads to a potentially increased search error.

Another way of integrating recurrent language models is to store the recurrent hidden state \( h(e_{i-1}) \) (c.f. Equation 4.8) in the search state, in which case the state is represented by \( (C, \hat{e}, j, h(e_{i-1})) \). However, such extension poses the same problem for recombination, since the hidden states are continuous-valued vectors likely to be different if the underlying target sequences they represent are not identical. To alleviate this problem, [Auli & Galley + 13] proposed to store the hidden states in the search states but to ignore them when comparing search states to decide on equivalency. State recombination will then not be affected, however, the recurrent language model scores become approximate, since only the hidden state corresponding to the best path is kept when recombining the nodes. This means that the target hypotheses that differ in the early history are treated as if they have identical hidden states. This is unlikely to be the case. However, the underlying assumption behind this approximation is that hidden states are mostly affected by the recent words used to compute them, and less affected by distant words. The authors proposed to store multiple hidden states per search state, but they found in experiments that storing one hidden state per search state is sufficient.

Instead of storing the recurrent hidden states in the search nodes, [Huang & Zweig + 14] proposed an alternative approach applied in the speech recognition domain. To allow sharing between nodes, a global cache is used to store the hidden states as values. The cache is accessed using the most recent \( n \) words in the target history used to generate its corresponding hidden state.

[Alkhouli & Rietig + 15] applied this method to phrase-based machine translation. Furthermore, they proposed to store the truncated history in the search state. The recurrent language model truncated history is not required to be the same as the \( n \)-gram language model history. Instead, the authors experimented with the effect of having different truncation lengths. We refer to the truncation length \( m \) as the \textit{caching order}. The truncated key is ignored during search state recombination. Storing the truncated history allows to control the trade-off between accuracy, speed and memory footprint—more accurate recurrent language model scores are obtained using a high caching order, but this results in slower evaluation as it implies fewer cache hits, and requires more memory to store the hidden states. On the other hand, a low caching order has less accurate neural scores, but it leads to more cache hits, which increases the overall translation speed and requires less memory. Note that even for high caching orders, there is still an approximation error due to recombination.

Consider the recurrent neural language model described in Equations 4.7-4.12. Assuming a standard recurrent layer, if \( E \) is the embedding size, and \( S_h \) is the hidden layer dimension, computing one step of the recurrent language model has a complexity of
\[
O\left( \frac{S_h^2 + S_h \cdot E + S_h \cdot |C|}{\text{recurrent layer}} + \frac{S_h \cdot \frac{|V_c|}{|C|}}{\text{class layer}} + \frac{V_c}{\text{word layer}} \right)
\]

We use three caches in decoding:

- The **hidden state cache** \( C_{\text{state}} \) is accessed by the key \( e_{i-m}^{i-1} \). It stores the hidden state \( h(e_{i-m}^{i-1}) \). The first time the target history \( e_{i-m}^{i-1} \) is encountered in search, the hidden state \( h(e_{1}^{i-1}) \) is computed using Equation 4.8 and stored in the cache. Every time the same history is encountered the state is looked up from the cache, saving redundant computations. In case of a hit, the time complexity is reduced by

\[
O(S_h^2 + S_h \cdot E).
\]

- The **normalization factor cache** \( C_{\text{norm}} \) is accessed by the key \( e_{i-m}^{i-1} \). It stores the normalization factor of the output layer that corresponds to the context \( e_{i-m}^{i-1} \). Since we use a class-factored output layer the cache stores the normalization factor of the class layer. The first time the target history \( e_{i-m}^{i-1} \) is encountered during search, the class normalization factor which is the denominator of the softmax function is computed using Equation 4.5. The normalization sum requires all raw class layer values computed using Equation 4.9. The \( C_{\text{norm}} \) cache eliminates redundant computations of Equation 4.9. A cache hit reduces the time complexity of the class layer from \( O(S_h \cdot |\mathcal{C}|) \) to \( O(S_h) \). We can use another cache to store the normalization factors of the word layer which requires using the key \( (c(e_i), e_{i-m}^{i-1}) \) (c.f. Equation 4.10). This cache however is hit less frequently than \( C_{\text{norm}} \) as it requires matching the word class in addition to the target history. A cache hit reduces the time complexity of the word layer from \( O(S_h \cdot |V_c|/|\mathcal{C}|) \) to \( O(S_h) \). Computing one time step of the recurrent language model then has a complexity of \( O(S_h) \)

- The **word probability cache** \( C_{\text{prob}} \) is accessed by the key \( (e_i, e_{i-m}^{i-1}) \). It stores the probability of word \( e_i \) conditioned on some target history ending with \( e_{i-m}^{i-1} \). This cache is useful for the case of node expansion in phrase-based search, where many expanding nodes may share the first part of the new phrase and differ only in the last part of the phrase. A cache hit leads to a time complexity of \( O(1) \).

The access order of these caches is as follows: first, the word probability is looked up using \( C_{\text{prob}}(e_i, e_{i-m}^{i-1}) \) and used if it is found in the cache. If it is not found, the hidden state corresponding to \( e_{i-m}^{i-1} \) is retrieved from the cache \( C_{\text{state}} \), and the class-factored output layer is computed using it. Note that \( C_{\text{state}}(e_{i-m}^{i-1}) \) always exists, as the target hypothesis is generated monotonically from left to right, and \( e_{i-m}^{i-1} \) will have already been processed. During the output layer computation, the normalization factor of \( e_{i-m}^{i-1} \) is looked up in the \( C_{\text{norm}} \) cache if it exists. The caches \( C_{\text{state}} \), \( C_{\text{norm}} \), and \( C_{\text{prob}} \) are updated with the newly obtained results.

Algorithm 5 shows the details of cache access order. It corresponds to the Equations 4.7–4.12, and uses three caches: \( C_{\text{state}}, C_{\text{norm}}, \) and \( C_{\text{prob}} \). The function \textsc{RnnlmProb} is called for each word probability evaluation \( p(e_i|e_{1}^{i-1}) \). Instead of using \( e_{1}^{i-1} \), we provide the last \( m \) words \( e_{i-m}^{i-1} \). The previous hidden state is retrieved in line 2, which should exist in the cache \( C_{\text{state}} \), as we assume monotone generation of the target hypotheses. If the new \( n \)-gram \( e_{i-m+1}^{i} \) has been computed before, the new hidden state is not recomputed (line 3); otherwise, the hidden state is advanced and the new entry is stored in the cache (lines 4–5). \textsc{AdvanceHiddenState} computes Equations 4.7–4.8. Line 7 checks for a cache hit in \( C_{\text{prob}} \) to skip the output layer computation. In line 11, \( C_{\text{norm}} \) is checked for a previously computed normalization factor. \( C_{\text{norm}} \) and \( C_{\text{prob}} \) are updated in lines 16–17. The class normalization factor is computed in line 27.
6.5 Neural Network Integration

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{RnnlmProb}(e_i, e_{i-1})</td>
<td>Returns the probability of generating ( e_i ) given ( e_{i-1} )</td>
</tr>
<tr>
<td>\text{AdvanceHiddenState}(e, h_{old})</td>
<td>Advances the hidden state to ( h_{new} )</td>
</tr>
<tr>
<td>\text{ClassProbability}(e, h, N)</td>
<td>Returns the class probability of ( e ) given ( h ) and ( N )</td>
</tr>
<tr>
<td>\text{WordProbability}(e, h)</td>
<td>Returns the word probability of ( e ) given ( h )</td>
</tr>
</tbody>
</table>

Algorithm 5: Cache access order for evaluating a recurrent language model with one recurrent layer and a class-factored output layer.

6.5.2 Recurrent Neural Network Lexical Model

The recurrent neural network lexical model described in Section 4.5.5 is similar to the recurrent language model in structure. The only exception is that it receives source input at each time step in addition to the target input. The model pairs each target word \( e_{i-1} \) with the next source word \( f_{b_i} \) to be translated. Due to the recurrent hidden layer, the hidden state encodes the full target history \( e_{1:i-1} \) and the full aligned source history \( f_{b_1:b_i} \). We follow the same caching strategy discussed in the previous section to integrate the recurrent lexical model into phrase-based search. To account for the extra source dependence, the target history \( e_{1:i-1} \) is augmented with the aligned source history to become \( (e_{1:i-1}, f_{b_1:b_i}) \). This key is used to access \( C_{\text{state}} \) and \( C_{\text{norm}} \), and
Algorithm 6: Cache access order for evaluating a recurrent lexical model with one recurrent layer and a class-factored output layer.

the key \((e_i, [e_{i-1}, f_{b_i}]_{i=i-m+1})\) is used to access \(C_{\text{prob}}\). Note the source is advanced by one step in comparison to the target history, since the translation model predicts the translation \(e_i\) aligned to the word \(f_{b_i}\); therefore, it is included as part of the source context.

Since the model assumes each target word to be paired with a source word, unaligned target words are aligned to a special token \(e_{\text{unaligned}}\) that is injected for each unaligned target word in the source side. It is injected next to the source word aligned to the most recent target word, preferring left to right [Sundermeyer & Alkhouri14]. An example is shown in Figure 6.5. For multiple-aligned source words, the most probable target word according to the source-to-target and target-to-source IBM 1 models is aligned to the source word, and the remaining target words are aligned to a special token \(e_{\text{aligned}}\) injected in the source side. Unaligned source words are ignored, and target words that are multiple-aligned are aligned to the most probable source word. That is, we do not inject special tokens in the target side. This is because determining the position for injections in the target might require access to a neighboring phrase that is not yet hypothesized; therefore, we only inject tokens in the source side. Note that since target phrase generation is monotonous, the previous phrase will always exist, and resolving alignment ambiguities from left to right will not encounter such issues.

Algorithm 6 presents the decoder integration of the recurrent lexical model. It is based on the recurrent language model integration algorithm. Due to the dependence on the source \(f_{b_i}\), the
6.5 Neural Network Integration

Figure 6.5: Illustration of word alignment, where alignment is indicated by the shaded areas. The alignment shown in Figure (a) is converted to the alignment shown in Figure (b). The special token $\epsilon_{\text{unaligned}}$ is associated with unaligned target words, and the special token $\epsilon_{\text{aligned}}$ is created when resolving multiple-aligned source words. The tokens are inserted in the source sentence, while the target sentence is left unchanged. Note that the source word ‘zur’ (marked in red) gets omitted in this conversion to obtain the one-to-one correspondence.

cache entry $C_{\text{state}}([e_{i-1}, f_i]_{i=m+1})$ may not already exist when the function is called; therefore, the function $\text{RNNLEXICALPROB}$ receives the extra word pair $(e_{i-m-1}, f_{b-i-m})$ as input to compute the hidden state if it is missing in the cache (lines 2–7). This has no effect on the cache key which is still accessed using $([e_{i-1}, f_i]_{i=m+1})$.

6.5.3 Bidirectional Recurrent Neural Network Lexical Model

Next, we discuss the integration of the bidirectional lexical model $p(e_i|f^T_i, b_i)$ given by Equation 4.28. This model is only conditioned on the current alignment point $b_i$ and the source sentence; therefore, its integration is less complicated than the previous models that depend on the target history. Since $b_i \in \{1...J\}$, there are at most $J$ distinct values for $g_i$ (c.f. Equation 4.27). Algorithm 7 shows how caching is applied in this case. During decoding, we compute Equations 4.22–4.27 one time for each alignment position that is hypothesized (line 10). If this part of the network is already computed (line 6), we skip it and only compute the output layer lines (13–14). The caches $C_{\text{state}}$, and $C_{\text{norm}}$ use the alignment $b_i$ as a key, and therefore they store at most $J$ distinct values each. We store word probability values in the cache $C_{\text{prob}}(e_i, b_j)$, which is accessed by a key composed of the target word and its aligned source position. Note that unlike the previous models that are conditioned on the target history, this model does not require storing extra information in the search nodes during decoding. This model can be computed exactly during decoding without any approximation, unlike the recurrent language and lexical models discussed above. Another alternative for computing the class layer is to compute the full normalized class probabilities in one matrix operation and look up the class probabilities when they are needed. We use this in our experiments.
The source and target sentences are pre-processed to have a one-to-one correspondence as follows: each unaligned source word is aligned to a special token $\epsilon_{unaligned}$ injected in the target sentence. The injection position is inherited from the alignment of the nearest source word preferring left to right. In the case of a multiple-aligned target word, the most likely source word according to IBM 1 models is aligned to it, and each of the remaining source words are aligned to a special token $\epsilon_{aligned}$ inserted in the target side. We ignore the unaligned target words and do not insert special tokens in the source sentence, so as to keep the bidirectional source representation independent of the alignment information. We also resolve multiple-aligned source words by assigning the source word to the most likely target word according to the IBM 1 model scores. The remaining target words are left unaligned. Note that due to the bidirectional source representation computed using the full sequence $f_{b1}$, the unaligned source words are included when computing the source representations. The resulting one-to-one correspondence can be contrasted to the one used for the recurrent lexical model discussed in Section 6.5.2. Instead of inserting special tokens in the source sentence, the bidirectional model uses sentences that have special tokens inserted in the target sentence.

### 6.6 $N$-Best List Rescoring

Direct use of neural models in phrase-based search allows scoring search nodes at each decoding step directly. Only the nodes with the best scores are kept during search, and the decision of which nodes are kept is influenced by the neural network models. Another approach to using neural networks in phrase-based systems is $N$-best rescoring, where a fixed list of translation hypotheses is first generated using the phrase-based system without using neural scores. In a second pass, the neural network scores are computed for the hypotheses, and the scores are combined with the phrase-based systems scores. The best translation is chosen as the one with the best combined score. A list of a maximum size of $N$ is generated for each source sentence, hence the name $N$-best
rescoring. The neural scores do not have any influence on the search regions navigated in the first pass. Moreover, the list represents a very small region of the search space, and therefore most of the search space explored in the first pass is not subject to neural scoring. Although this is suboptimal, N-best rescoring has its advantages. First, it can be more efficient to score a limited list of hypotheses instead of scoring the search nodes directly during search, since there is a large number of search nodes that need to be scored. Second, some models cannot be used directly in decoding, such as the bidirectional lexical models that pair the source and target word embeddings using the alignment information (c.f. Section 4.5.6). N-best lists contain the complete information \((e_1^f, b_1^f, f_1^f)\). Since the full alignment information is available, the bidirectional lexical model with paired embeddings can be applied in N-best rescoring, but not directly in phrase-based decoding.

In Section 6.7, we present results for applying neural models to score N-best lists and compare that to direct neural model integration in search.

### 6.7 Experimental Evaluation

In this section, we present results for the integration of the neural language and translation models into phrase-based search, and compare the results to the second pass N-best list neural rescoring. We carry out the phrase-based experiments in *Jane*, which is RWTH Aachen University’s open-source toolkit for statistical machine translation. It is described in detail in [Vilar & Stein\(^+\) 10, Stein & Vilar\(^+\) 11, Wuebker & Huck\(^+\) 12]. We extend the framework to integrate neural models into search. The toolkit is mostly written in C++. We use an extension of *rathlm* [Sundermeyer & Schlüter\(^+\) 14] to train all the neural networks used for phrase-based experiments. We also extended the toolkit to score N-best lists generated by *Jane*.

#### 6.7.1 Setup

We carry out experiments on the IWSLT 2013 German→English shared translation task.\(^1\) The corpora statistics are given in Appendix A.1. The baseline system is trained on all available bilingual data, 4.3M sentence pairs in total, and uses a 4-gram LM with modified Kneser-Ney smoothing [Kneser & Ney 95, Chen & Goodman 98], trained with the SRILM toolkit [Stolcke 02]. As additional data sources for the LM, we selected parts of the Shuffled News and LDC English Gigaword corpora based on the cross-entropy difference [Moore & Lewis 10], resulting in a total of 1.7 billion running words for LM training. The baseline is a standard phrase-based SMT system [Koehn & Och\(^+\) 03] tuned with MERT [Och 03]. It contains a hierarchical reordering model [Galley & Manning 08], a 7-gram word cluster language model [Wuebker & Peitz\(^+\) 13b], and three phrase-count indicator models. All lexical neural networks are trained on the TED portion of the data of 138K segments (cf. Table A.1), and the recurrent neural language model is trained on 159K segments (cf. Table A.2). The experiments are run using an observation histogram size of 100, with a maximum of 16 lexical hypotheses per source coverage and a maximum of 32 reordering alternatives per source cardinality.

Additional experiments are performed on the Arabic→English task of the DARPA BOLT project. The phrase-based system contains a hierarchical reordering model and three phrase-count indicator models. The corpora statistics are given in Appendix A.2.2. The system is a standard phrase-based decoder trained on 922K segments, amounting to 15.5M English running words, and using 17 dense features. The neural network training is performed using the same data. We evaluate results on two data sets from the ‘discussion forum’ domain, DEV12-Dev and P1R6-Dev. The development set DEV12-Tune is used to tune the phrase-based system. Both DEV12-tune and P1R6-tune are used to train the neural networks. The experiments are run

\(^1\)http://www.iwslt2013.org
Table 6.1: IWSLT 2013 German→English results. We use the caching order \( n = 8 \) for the language model experiments and \( n = 5 \) for the unidirectional lexical model experiments. The bidirectional lexical model is computed exactly.

<table>
<thead>
<tr>
<th>phrase-based baseline</th>
<th>-</th>
<th>33.4</th>
<th>46.1</th>
<th>30.6</th>
<th>49.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ recurrent language model</td>
<td>rescoring</td>
<td>34.1</td>
<td>45.7</td>
<td>31.5</td>
<td>48.6</td>
</tr>
<tr>
<td></td>
<td>decoding</td>
<td>33.9</td>
<td>45.7</td>
<td>31.6</td>
<td>48.3</td>
</tr>
<tr>
<td>+ bidirectional lexical model</td>
<td>rescoring</td>
<td>34.4</td>
<td>45.3</td>
<td>32.2</td>
<td>47.8</td>
</tr>
<tr>
<td></td>
<td>decoding</td>
<td>34.4</td>
<td>44.9</td>
<td>32.3</td>
<td>47.3</td>
</tr>
<tr>
<td>+ unidirectional lexical model</td>
<td>rescoring</td>
<td>34.3</td>
<td>45.4</td>
<td>31.6</td>
<td>48.3</td>
</tr>
<tr>
<td></td>
<td>decoding</td>
<td>34.4</td>
<td>45.6</td>
<td>31.6</td>
<td>48.2</td>
</tr>
</tbody>
</table>

using an observation histogram size of 100, with a maximum of 32 lexical hypotheses per source coverage and a maximum of 8 reordering alternatives per source cardinality.

We also perform experiments on the Chinese→English DARPA BOLT task. The phrase-based system contains a hierarchical reordering model, a 7-gram word cluster language model, and the enhanced low-frequency model. The corpora statistics are given in Appendix A.2.1. The task has 4.08M training segments (cf. Table A.4). It also has a small training set from the discussion forum domain. The in-domain data has 64.8K segments which we used to train the lexical neural networks (cf. Table A.5). The recurrent language model is trained on a super set of 354K segments including the in-domain target data, amounting to 8.6M running English words. The full details are given in Table A.6. We tune the phrase-based system and the neural networks on DEV12-Tune and use DEV12-Dev and P1R6-dev as blind test sets for evaluation.

We experiment with the bidirectional lexical model which has two bidirectional LSTM layers and another LSTM layer. The model is described in Equations 4.13–4.20. We also experiment with the unidirectional lexical model of Section 4.5.5, which has one LSTM layer, and includes the target information as input. The recurrent language model used in the section is described in Section 4.5.4, and it has one LSTM layer. We use word embeddings of size 200. All LSTM layers have 200 nodes. We use a class-factored output layer of 2000 classes, 1000 classes are dedicated to the most frequent 1000 words, where each word is mapped to its unique class, and the remaining 1000 classes are shared among the rest of the vocabulary. The neural networks are trained using stochastic gradient descent. The learning rate is halved each time there is a perplexity increase on the development set. We discard the epoch that has a perplexity increase and reset the model to the last weights that resulted in perplexity decrease. The development perplexity is checked after each epoch.

All results are measured in case-insensitive BLEU [%] [Papineni & Roukos+ 02] and TER [%] [Snover & Dorr+ 06] on a single reference. Rescoring experiments are performed using 1000-best lists (without duplicates), where an additional MERT iteration is performed. 20 such trials are carried out and the average results are reported. We used the multeval toolkit [Clark & Dyer+ 11] for evaluation.
6.7 Experimental Evaluation

6.7.2 Translation Quality

We first conduct experiments to compare the effect of integrating neural language and lexical models directly in phrase-based search, and compare that to second pass N-best rescoring. The results on IWSLT 2013 German→English task are shown in Table 6.1. We observe the following:

- Each of the neural models improve over the baseline, whether they are applied in rescoring or decoding.
- The bidirectional lexical model, which uses the full source sentence but no target history, outperforms the unidirectional lexical model, which uses the target history and partial source information. This suggests that using the full source information is more important than using the target history. In principle, the target history is generated using the source information only, so the model does not obtain extra information when including the target history. Since these systems include a separate language model, this can justify the observation that including the target history is not as important when the system has other knowledge sources integrated that cover target language modeling. However, we expect this to be an issue if the model is to be used standalone to generate translation as in neural machine translation.
- The comparison between rescoring and decoding for each of the models shows that decoding might have a slight advantage over rescoring, but the difference is minimal.
- Although the bidirectional lexical model is computed exactly in decoding, there is a slight advantage for using it in decoding compared to rescoring, which can be attributed to a more informed selection of beam entries during decoding, allowing search to reach hypotheses not reached by the baseline system, which are not accessible for rescoring that is applied to the N-best lists of the baseline system.

Table 6.2 shows the results of the BOLT Arabic→English task. We observe the following:

- There are similar trends compared to the previous task, with consistent improvements using the neural models whether they are applied in decoding or rescoring.
- The difference between the bidirectional and unidirectional lexical models is smaller. We note that the models are trained on more parallel data compared to the previous German→English task. This can be useful in training the unidirectional lexical model that pairs source and target information on the embedding level, something that might require more training data to get the best out of the model.
- There is no clear winner when comparing decoding and rescoring.

The BOLT Chinese→English results are shown in Table 6.3. In light of these results, we note the following:

- All models consistently improve the baseline.
- Decoding results are between 0.2-0.6 BLEU better than rescoring on the test sets DEV12-Dev and P1R6-Dev. We note that a small amount of training data is used to train the lexical models which have larger discrepancy between rescoring and decoding compared to the recurrent language model. The lexical models are trained on 1.7M English tokens, whereas the language models are trained on 8.6M tokens. The discrepancy for the language model is smaller.

We conclude that integrating all models improved the baselines. N-best rescoring results are close to decoding results when the amount of training data used to train the models is not too small.
6 Phrase-Based Machine Translation

Table 6.2: BOLT Arabic→English results. We use the caching order \( n = 8 \) for the language model experiments and \( n = 10 \) for the unidirectional lexical model experiments. The bidirectional lexical model is computed exactly.

<table>
<thead>
<tr>
<th>System</th>
<th>scoring method</th>
<th>DEV12-tune</th>
<th>DEV12-dev</th>
<th>P1R6-dev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BLEU [%]</td>
<td>TER [%]</td>
<td>BLEU [%]</td>
</tr>
<tr>
<td>phrase-based baseline</td>
<td>-</td>
<td>27.1 58.2</td>
<td>23.9 59.7</td>
<td>26.4 59.8</td>
</tr>
<tr>
<td>+ recurrent language model</td>
<td>rescoring</td>
<td>27.3 58.0</td>
<td>24.3 59.3</td>
<td>26.9 59.3</td>
</tr>
<tr>
<td></td>
<td>decoding</td>
<td>27.4 57.7</td>
<td>24.6 59.0</td>
<td>27.0 59.2</td>
</tr>
<tr>
<td>+ bidirectional lexical model</td>
<td>rescoring</td>
<td>27.7 57.4</td>
<td>24.7 58.9</td>
<td>27.0 58.9</td>
</tr>
<tr>
<td></td>
<td>decoding</td>
<td>27.8 57.5</td>
<td>24.8 58.9</td>
<td>27.0 58.9</td>
</tr>
<tr>
<td>+ unidirectional lexical model</td>
<td>rescoring</td>
<td>27.3 57.8</td>
<td>24.4 59.0</td>
<td>27.2 59.0</td>
</tr>
<tr>
<td></td>
<td>decoding</td>
<td>27.4 57.7</td>
<td>24.5 59.0</td>
<td>27.3 59.0</td>
</tr>
</tbody>
</table>

Table 6.3: BOLT Chinese→English results. We use the caching order \( n = 8 \) for the language model experiments and \( n = 5 \) for the unidirectional lexical model experiments. The bidirectional lexical model is computed exactly.

<table>
<thead>
<tr>
<th>System</th>
<th>scoring method</th>
<th>DEV12-tune</th>
<th>DEV12-dev</th>
<th>P1R6-dev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BLEU [%]</td>
<td>TER [%]</td>
<td>BLEU [%]</td>
</tr>
<tr>
<td>phrase-based baseline</td>
<td>-</td>
<td>20.1 66.0</td>
<td>18.2 68.2</td>
<td>17.4 66.9</td>
</tr>
<tr>
<td>+ recurrent language model</td>
<td>rescoring</td>
<td>20.7 65.2</td>
<td>18.5 67.3</td>
<td>18.0 66.5</td>
</tr>
<tr>
<td></td>
<td>decoding</td>
<td>20.7 65.2</td>
<td>18.7 67.4</td>
<td>18.2 66.2</td>
</tr>
<tr>
<td>+ bidirectional lexical model</td>
<td>rescoring</td>
<td>20.3 65.5</td>
<td>18.4 67.7</td>
<td>17.6 66.8</td>
</tr>
<tr>
<td></td>
<td>decoding</td>
<td>20.6 65.1</td>
<td>18.7 67.5</td>
<td>17.9 66.8</td>
</tr>
<tr>
<td>+ unidirectional lexical model</td>
<td>rescoring</td>
<td>20.2 65.4</td>
<td>18.1 67.4</td>
<td>17.4 66.4</td>
</tr>
<tr>
<td></td>
<td>decoding</td>
<td>20.4 65.2</td>
<td>18.6 67.5</td>
<td>18.0 66.3</td>
</tr>
</tbody>
</table>

6.7.3 Approximation Analysis

The integration of the recurrent models that include target history into phrase-based search is approximate. The neural model scores are not computed exactly to allow for sufficient beam diversity (cf. Sections 6.5.2, 6.5.1). To study the effect of approximation, we generated \( N \)-best lists using the approximate integration method. We then computed exact neural scores of the \( N \)-best list entries. The results are shown in Table 6.4. Computing exact scores in rescoring improves the results consistently, but also marginally on both German→English and Arabic→English tasks. This indicates that the approximation effect is minimal.

Even for a high caching order, there is an approximation error due to recombination. We analyze the recombination error by storing the exact cache key without truncation in the search node, isolating recombination as the only source of error. We generate 1000-best lists using approximate recurrent language model scores during decoding. Afterwards, we compute the exact RNN scores of the 1000-best lists and compare them to the approximate scores. Figure 6.6 shows the cumulative distribution of the absolute relative difference between the approximate and exact model scores divided by the true score. The results are computed for the IWSLT German→English task. While the difference is non-zero for most sentences in the \( N \)-best lists
6.7 Experimental Evaluation

Table 6.4: N-best list rescoring results after first-pass decoding including the same neural model. The neural model scores are computed with approximation during decoding, but they are computed exactly in the second pass. The results are reported on the development sets. The decoding caching order for the language model and lexical model experiments is 8 and 10, respectively.

<table>
<thead>
<tr>
<th>model</th>
<th>scoring method</th>
<th>IWSLT</th>
<th>BOLT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>German→English</td>
<td>Arabic→English</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BLEU [%]</td>
<td>BLEU [%]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TER [%]</td>
<td>TER [%]</td>
</tr>
<tr>
<td>recurrent language model</td>
<td>decoding</td>
<td>33.9</td>
<td>27.4</td>
</tr>
<tr>
<td></td>
<td>decoding + rescoring</td>
<td>34.1</td>
<td>27.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>45.7</td>
<td>57.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>45.8</td>
<td>57.6</td>
</tr>
<tr>
<td>unidirectional lexical model</td>
<td>decoding</td>
<td>34.4</td>
<td>27.4</td>
</tr>
<tr>
<td></td>
<td>decoding + rescoring</td>
<td>34.6</td>
<td>27.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>45.6</td>
<td>57.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>45.3</td>
<td>57.7</td>
</tr>
</tbody>
</table>

Figure 6.6: The cumulative distribution of the absolute relative difference between the approximate and true RNN score with respect to the true score. The distribution was generated using around 233k sentences, obtained from n-best lists generated by decoding the dev set of the IWSLT task.

used to generate this figure, 66% of the cases have a relative error of 1% at most, and the error is 6% at most for 99% of the cases.

The caching order is used as a key to access the global cache that stores the recurrent hidden states. We expect using lower caching orders to cause more cache hits. This is also expected to divert the approximate scores more from the exact scores. We varied the caching order on the IWSLT German→English task for the recurrent language model integration experiments and measured the corresponding translation quality. Table 6.5 shows that the caching order does have an impact on translation quality as expected. Setting the caching order to 2 and 4 results in deterioration in translation quality. Setting the caching order to 8 at least gives the best results. The last row shows the case of unlimited caching order, i.e. each unique target sequence keeps track of its hidden states. This leads to exact score computation, but it also decreases the hit ratio, and slows down decoding significantly as we will show in Section 6.7.4.
Table 6.5: A comparison between storing the recurrent hidden state in the search nodes (last entry) or caching it using different caching orders (remaining entries). We report the BLEU [%] scores for the IWSLT 2013 German→English task.

<table>
<thead>
<tr>
<th>Caching Order</th>
<th>dev</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>33.1</td>
<td>30.8</td>
</tr>
<tr>
<td>4</td>
<td>33.4</td>
<td>31.2</td>
</tr>
<tr>
<td>6</td>
<td>33.9</td>
<td>31.6</td>
</tr>
<tr>
<td>8</td>
<td>33.9</td>
<td>31.5</td>
</tr>
<tr>
<td>16</td>
<td>34.0</td>
<td>31.5</td>
</tr>
<tr>
<td>30</td>
<td>33.9</td>
<td>31.5</td>
</tr>
</tbody>
</table>

Figure 6.7: A comparison between applying the recurrent language model in decoding using caching, and applying it exactly in rescoring. Note that the caching order is a parameter for decoding, and it is not related to rescoring.

Another way to analyze the effect of approximation is to compare the search scores of the winning hypotheses. In one setup, we compute approximate recurrent language model scores directly during decoding. In the contrastive setup, we generate 500-best lists using the baseline phrase-based system that does not include the recurrent language model, and compute exact recurrent model scores for them offline after decoding, we then re-rank the lists according to the new search scores that include the recurrent language model. We use the same set of log-linear weights in both setups to factor out the MERT optimization error and focus on the search error. We compare the search scores of the winning hypotheses from each setup and mark a win for the method that has a better search score. Figure 6.7 shows the results computed for different caching order values. We ignore tie cases where both methods agree on the hypothesis. We observe that for low caching orders, N-best rescoring yields more hypotheses with better search scores compared to integrated decoding, which is due to the highly inaccurate model scores computed during decoding with such caching orders. Only starting from a caching order of 6 does integrated neural decoding have better search scores. With high caching orders, decoding has better search scores in 35% of the sentences, compared to 12% for N-best rescoring, the remaining 53% are tie cases. This means for 88% of the cases, integrated decoding is at least as good as rescoring in terms of finding better search scores.
6.7.4 Caching Analysis

We investigate the hit ratio for the different cache types introduced in Section 6.5.1. Figure 6.8 shows the percentage of cache hits for different caching orders. We count a cache hit if a look up is performed on that cache and the entry is found, otherwise the look up counts as a cache miss. We observe high hit ratios even for high caching orders. This is due to the fact that most of the hits occur upon node expansion, where a node is extended by a new phrase, and where all candidates share the same history. We also observe that word probabilities are retrieved from the cache 70% of the time for high enough caching orders, which happens when hypothesizing phrase candidates that have similar beginnings. Note also that the reported $C_{norm}$ hit ratio is for the cases where the cache $C_{prob}$ produces a cache miss. We report this hit ratio since the original $C_{norm}$ hit ratio is equal to $C_{state}$’s hit ratio as they both use the same caching key.

Using a sentence-level global cache reduces redundant computation during search. We measure the speed up due to using the three cache types. The results are shown in Table 6.6. We reduce the approximation effect by setting the caching order to a large value of 30 when caching is used. All rows have the same translation quality. Caching the recurrent hidden state improves the decoding speed, but most of the improvement is due to the $C_{norm}$ cache, which caches the normalization factor of the class layer. Even though we are using a class-factored output layer to reduce the size of the vocabulary used to compute the normalization factor (2000 classes here), it is necessary to cache the class normalization factor, which brings about a four-fold speed-up. Using the word probability cache $C_{prob}$ has negligible effect on speed, due to the small matrices used to compute the word layer part, thanks to class-factorization. Overall, compared to the baseline, a speed-up factor of 6.3 is achieved when all caches are used.

6.8 Contributions

The author adapted the ideas of [Huang & Zweig+ 14] from the speech recognition domain, and applied it to integrate recurrent language and translation models into phrase-based decoding. [Huang & Zweig+ 14] introduce a global cache to store the hidden states as values. The cache is accessed using a key consisting of the most recent $n$ target words. The author proposed in
Table 6.6: The effect of using caching on translation speed. A large caching order of 30 is used to reduce the approximation effect of caching, leading to the same translation quality for all table entries.

<table>
<thead>
<tr>
<th>Cache</th>
<th>Speed [words/second]</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>0.03</td>
</tr>
<tr>
<td>$C_{\text{state}}$</td>
<td>0.05</td>
</tr>
<tr>
<td>$C_{\text{state}} + C_{\text{norm}}$</td>
<td>0.19</td>
</tr>
<tr>
<td>$C_{\text{state}} + C_{\text{norm}} + C_{\text{prob}}$</td>
<td>0.19</td>
</tr>
</tbody>
</table>

[Alkhouli & Rietig 15] to apply this technique to recurrent neural integration into phrase-based decoding. Furthermore, the work introduces the caching order concept to store the truncated history in the search state, which can be different to size of the language model history. This allows to control the trade-off between accuracy, speed and memory footprint. Felix Rietig implemented caching for the recurrent neural network language described in Section 6.5.1, and the recurrent neural network lexicon model described in Section 6.5.2. He implemented his code into Jane and rwthlm during his Master thesis, which was supervised by the author. The bidirectional recurrent neural network lexicon model of Section 6.5.3 is proposed and implemented into Jane and rwthlm by the author. The author implemented $N$-best rescoring described in Section 6.6 into rwthlm. The rescoring tool is used in [Sundermeyer & Alkhouli 14] to rescore $N$-best lists using recurrent language models, recurrent lexicon models and bidirectional recurrent lexicon models. The author designed and implemented the bidirectional models published in [Sundermeyer & Alkhouli 14], which are used to rescore $N$-best lists of phrase-based systems.
7. Conclusion and Scientific Achievements

We will list the scientific achievements of this work in relation to the scientific goals introduced in Chapter 2.

• We presented different novel architectures for alignment-based neural networks in Chapter 4. We introduced a novel feedforward alignment model in Section 4.5.2. We also introduced bidirectional RNN lexicon models in Section 4.5.6, with variants that are autoregressive, i.e. including the target history, and non-autoregressive, that do not depend on the target history. We presented different ways of combining the target and source words within the model, keeping in mind the use of the model in search. Similarly, we introduced a bidirectional alignment model in Section 4.5.7. While these models do not include an attention component, we also introduced hybrid models in Section 4.7 that mix alignment and attention. A recurrent model with attention and alignment bias was introduced in Section 4.7.1. Alignment heads combined with the multi-attention array of the transformer model was introduced in Section 4.7.2.

• We compared the performance of pure alignment-based recurrent neural machine translation systems to the standard phrase-based, RNN attention, and transformer baselines (Section 5.7.2). We also compared the performance of hybrid RNN attention and transformer systems that include alignment. We observed that adding alignment to the RNN attention improves performance on the English→Romanian and Chinese→English tasks. We could not improve the German→English RNN attention model by adding alignment, but that did not result in any significant degradation either. We also observed that adding alignment to the transformer maintains its performance. In other words, using alignment at least maintains the performance of RNN attention and transformer systems.

• We compared pure feedforward and RNN alignment-based systems in Section 5.7.4, and found that RNN systems outperformed feedforward systems, which is not surprising given that RNN models capture more source and target context. Note that the transformer architecture overcomes the limited context by employing self-attention, which accesses the full source and target context. We also compared using class-factored output layers to 50k-BPE layers and found that, on large tasks, using subword vocabularies is preferable in terms of performance, where we also observe 0% OOV rate on the target side.

• We investigated speeding up decoding in alignment-based neural machine translation in two ways. In Section 5.7.7, we investigated using a class-factored output to speed up decoding. We demonstrated that the use of class-factored output can speed up decoding, without loss of translation quality, by limiting translation candidates to the top scoring classes. We also introduced alignment pruning in Section 5.7.10 and showed how to use it to speed up alignment-based systems.
7 Conclusion and Scientific Achievements

- We studied the alignment quality in Section 5.7.11 and demonstrated that adding alignment to RNN attention and transformer systems yields significant improvements in terms of the alignment error rate. We also showed that these improvements could be observed in a dictionary suggestions task where an external dictionary is used to override translation (Section 5.7.12). Since detecting which words to override is crucial, alignment-based systems outperform the standard systems on such tasks.

- We studied the effect of using forced-alignment training on alignment-based systems in Section 5.7.13. We observed that, while forced-alignment training results in systems that have lower search cost, the improvements were only significant for the Chinese→English task. Since forced-alignment training is expensive, we only recommend it for low-performing tasks, and when long training time can be tolerated.

- We compared $N$-best rescoring to the direct integration of recurrent neural language and lexical models in phrase-based systems in Section 6.7.2. We found that using the models either way consistently improved the underlying phrase-based systems. Direct decoder integration achieves similar or superior performance to $N$-best rescoring. In direct decoder integration, the neural model directly influences the selection of phrases to use during beam search. In contrast, $N$-best rescoring limits the model contribution to selecting complete translation candidates from a list generated during standard phrase-based decoding, hence, the model influence is constrained. Nevertheless, the improvements achieved during $N$-best rescoring are large enough to justify using it, especially considering translation speed, which is much slower for directly integrated models. We recommend using $N$-best rescoring with recurrent neural models where speed is crucial, and recommend using the integration approach where translation quality is the more decisive factor.

7.1 Outlook

Neural machine translation has become the new paradigm to dominate the machine translation research and industrial markets during the course of this dissertation. The introduction of attention enabled end-to-end modeling, simplifying the steps required to build a machine translation system. Machine translation output has become clearly more fluent, making it more appealing to humans. Nevertheless, with the introduction of neural machine translation, there is less control over the system output in comparison to phrase-based systems. As the world grows more globalized, and in an era where social media platforms enable vast amount of free user-generated content to reach millions of users, machine translation plays an important role in faster dissemination of information across the globe. The rise of machine translation is but one among many of the artificial intelligence products that already made it to the market, such as face recognition, face filters, speech recognition in digital assistants, etc. As automation rapidly unrolls to the market, and algorithms are becoming more responsible for making decisions, important ethical question are raised, and a first step to answer them is to have explainable models. This thesis partly focused on re-introducing the alignment concept to neural machine translation, and showed that the proposed models help improve the alignment error rate, which implies knowing which source words generated which target words. We also demonstrated that introducing alignment facilitates translation override, giving the user more influence over the generated translation. While the alignment approach is successful, there is still more work to be done to bring these systems to a speed on par with the standard neural machine translation systems. Moreover, alignment can be helpful in designing streaming systems that are required to generate live output without waiting for the full input sentence.
8. Individual Contributions

In this chapter we list the individual contributions of the author in contrast to team work effort, putting what is described in Sections 4.9, 5.8, and 6.8 in one place.

The feedforward alignment model in Section 4.5.2 was proposed in [Alkhouli & Bretschner+ 16]. The author of this dissertation contributed the idea and the implementation of this model. The model was implemented in Jane. The model architecture is based on the feedforward lexicon model of [Devlin & Zbib+ 14]. The recurrent neural network lexicon model (Section 4.5.5) and the bidirectional recurrent lexicon model that pairs aligned source and target word embeddings (Section 4.5.6) were proposed in [Sundermeyer & Alkhouli+ 14]. Martin Sundermeyer and Jørn Wuebker contributed the unidirectional lexical model. Martin Sundermeyer proposed using bidirectional layers to capture the full source context. The author of this dissertation designed and implemented the bidirectional model in rwthlm. The author implemented N-best rescoring described in Section 6.6 in rwthlm. The rescoring tool was used in [Sundermeyer & Alkhouli+ 14] to rescore N-best lists using recurrent language models, recurrent lexicon models and bidirectional recurrent lexicon models. The author adapted the ideas of [Huang & Zweig+ 14] from the speech recognition domain, and applied it to integrate recurrent language and translation models into phrase-based decoding [Alkhouli & Rietig+ 15]. Furthermore, the paper introduced the caching order concept to store the truncated target history in the search state, which can be different to the size of the language model history. This allows controlling the trade-off between accuracy, speed, and memory footprint. Felix Rietig implemented caching for the RNN language model described in Section 6.5.1, and the recurrent neural network lexicon model described in Section 6.5.2. He implemented it in Jane and rwthlm during his master thesis, which was supervised by the author. The bidirectional recurrent neural network lexicon model of Section 6.5.3 was proposed and implemented in Jane and rwthlm by the author. The author proposed in this dissertation a bidirectional autoregressive lexicon model including recurrency over the target sequence. The author contributed the idea, model design, and implementation in the Sockeye toolkit. The model was implemented in Jane by Mohammad Hethnawi during his Master thesis, which was supervised by the author of this dissertation. All these variants were described in Section 4.5.6. Section 4.5.7 describes the bidirectional alignment model proposed in [Alkhouli & Ney 17]. The author contributed the idea, model design, and implementation in Sockeye. Mohammed Hethnawi implemented it in Jane. The paper also described the alignment-biased attention-based recurrent model (Section 4.7.1). Biasing the attention component using the alignment information was completely contributed by the author. The multi-head self-attentive lexicon and alignment models were proposed in [Alkhouli & Bretschner+ 18]. The models are described in Sections 4.7.2 and 4.7.3. The author proposed the idea and model design. Gabriel Bretschner implemented the models in Sockeye during his Master thesis, which was supervised by the author.

The author proposed the alignment-based decoding framework in [Alkhouli & Bretschner+ 16], where feedforward lexicon and feedforward alignment models were initially used. The search algorithm is an extension of the simple beam search algorithm used in [Bahdanau & Cho+ 14,
Sutskever & Vinyals+ 14, Cho & van Merrienboer+ 14a]. The author contributed the idea and the design of the search algorithm. The decoder was implemented by the author in *Sockeye*. Gabriel Bretschner implemented it in *Jane* as part of his bachelor thesis which was supervised by the author. The model combinations used in the alignment-based framework described in Section 5.6.1 are contributions of the author. The forced-alignment training algorithm was proposed by the author. The author implemented the initial version of forced-alignment training in *Jane*, which was later extended by Gabriel Bretschner during his bachelor thesis and Mohammed Hethnawi during his master thesis; both were supervised by the author. The author implemented the forced-alignment training framework that consists of the full pipeline described in Figure 5.2. All forced-alignment training experiments reported in this work were done using the *Jane* toolkit.

The author proposed the search pruning techniques of Section 5.6.4 in [Alkhouli & Bretschner+ 16, Alkhouli & Bretschner+ 18]. Gabriel Bretschner implemented class-based lexical pruning in *rwthlm* and alignment pruning in *Sockeye* during his bachelor and master theses. The soft and hard alignment extraction methods described in Section 5.6.5 are contributions of the author [Alkhouli & Ney 17, Alkhouli & Bretschner+ 18].
A. Corpora

A.1 International Workshop on Spoken Language Translation

The International Workshop on Spoken Language Translation (IWSLT) is an annual evaluation campaign that focuses on the translation of TED talks\(^1\). Data statistics for the IWSLT 2013\(^2\) shared task are shown in Table A.1. This includes statistics for all available training data and the TED talk data, which is the domain of the task. The development and test sets are also from the TED talk domain. The data used to train the recurrent language model are shown in Table A.2.

Table A.1: Corpus Statistics for the bilingual data provided for the IWSLT 2013 German→English task. The TED in-domain data statistics are also shown. The neural network vocabulary is extracted from this in-domain data by removing singletons from the vocabulary. Out-of-vocabulary (OOV) numbers refer to running words.

<table>
<thead>
<tr>
<th></th>
<th>German</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>train:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>4.3M</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>108M</td>
<td>109M</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>836K</td>
<td>792K</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>8.6K (8.0%)</td>
<td>7.1M (6.5%)</td>
</tr>
<tr>
<td>train (TED):</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>138K</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>2.6M</td>
<td>2.7M</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>75.4K</td>
<td>50.2K</td>
</tr>
<tr>
<td>Neural Network Vocabulary</td>
<td>40.7K</td>
<td>29.7K</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>34.7K (1.3%)</td>
<td>20.5K (0.8%)</td>
</tr>
<tr>
<td>dev:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>885</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>20K</td>
<td>19.3K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>4,000</td>
<td>3,161</td>
</tr>
<tr>
<td>OOVs</td>
<td>95 (0.5%)</td>
<td>82 (0.4%)</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>545 (2.4%)</td>
<td>374 (1.9%)</td>
</tr>
<tr>
<td>test:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1565</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>31.1K</td>
<td>32.0K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>4,949</td>
<td>3,703</td>
</tr>
<tr>
<td>OOVs</td>
<td>94 (0.3%)</td>
<td>68 (0.2%)</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>696 (2.2%)</td>
<td>411 (1.3%)</td>
</tr>
</tbody>
</table>

\(^1\)http://www.ted.com/talks
\(^2\)http://www.iwslt2013.org
Table A.2: Corpus statistics for the TED portion of the German→English training data used to train the recurrent neural language model. Out-of-vocabulary (OOV) numbers refer to running words. The neural network vocabulary is created by removing singleton words.

<table>
<thead>
<tr>
<th></th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>train (TED):</strong></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>159K</td>
</tr>
<tr>
<td>Running Words</td>
<td>3.2M</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>50K</td>
</tr>
<tr>
<td>Neural Network Vocabulary</td>
<td>32.5K</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>22.3K (0.7%)</td>
</tr>
<tr>
<td><strong>dev:</strong></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>885</td>
</tr>
<tr>
<td>Running Words</td>
<td>19.3K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>3,161</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>360 (1.9%)</td>
</tr>
<tr>
<td><strong>test:</strong></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1565</td>
</tr>
<tr>
<td>Running Words</td>
<td>32.0K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>3,703</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>911 (2.9%)</td>
</tr>
</tbody>
</table>

A.2 BOLT

The DARPA BOLT (Broad Operational Language Translation) Program aims to develop genre-independent machine translation systems. BOLT is focused on improving translation for informal genres with user-generated content such as discussion forums and chat.

A.2.1 BOLT Chinese→English

Table A.4 presents the corpus statistics for the Chinese→English BOLT task. DEV12-tune, DEV12-dev and P1R6-dev are taken from the discussion forum domain. The full data statistics used to train the phrase-based models are given in Table A.4. There is a small part of the training data that belongs to the discussion forum domain. The data statistics for this in-domain data is given in Table A.5. A super set containing the in-domain data is described in Table A.6. This set is used to train the recurrent language model.

A.2.2 BOLT Arabic→English

Table A.7 presents the corpus statistics for the Arabic→English BOLT task. We tune and evaluate for the discussion forum domain using the tuning set DEV12-tune and the test sets DEV12-dev and P1R6-dev.
Table A.3: Subword-level corpus statistics for the Chinese→English BOLT task. Out-of-vocabulary (OOV) numbers refer to running words. BPE subwords are created using 50k BPE merge operations. The source side has more than 50k words. We limit the source vocabulary to the most frequent 50k source subwords.

<table>
<thead>
<tr>
<th></th>
<th>Chinese</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>train:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>4.1M</td>
<td></td>
</tr>
<tr>
<td>Running Subwords</td>
<td>80.3M</td>
<td>87.7M</td>
</tr>
<tr>
<td>Subword Vocabulary</td>
<td>50.0K</td>
<td>49.7K</td>
</tr>
<tr>
<td><strong>DEV12-tune (dev):</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1842</td>
<td></td>
</tr>
<tr>
<td>Running Subwords</td>
<td>38.4K</td>
<td>47.0K</td>
</tr>
<tr>
<td>Subword Vocabulary</td>
<td>6,201</td>
<td>5,860</td>
</tr>
<tr>
<td>OOVs</td>
<td>94 (0.2%)</td>
<td>2 (0.0%)</td>
</tr>
<tr>
<td><strong>DEV12-dev:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1844</td>
<td></td>
</tr>
<tr>
<td>Running Subwords</td>
<td>38.9K</td>
<td>29.6K</td>
</tr>
<tr>
<td>Subword Vocabulary</td>
<td>5,928</td>
<td>3,885</td>
</tr>
<tr>
<td>OOVs</td>
<td>173 (0.4%)</td>
<td>9 (0.0%)</td>
</tr>
<tr>
<td><strong>P1R6-dev:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1124</td>
<td></td>
</tr>
<tr>
<td>Running Subwords</td>
<td>24.1K</td>
<td>49.4K</td>
</tr>
<tr>
<td>Subword Vocabulary</td>
<td>4,399</td>
<td>5,800</td>
</tr>
<tr>
<td>OOVs</td>
<td>117 (0.5%)</td>
<td>2 (0.0%)</td>
</tr>
</tbody>
</table>

Table A.4: Word-level corpus statistics for the Chinese→English BOLT task. Out-of-vocabulary (OOV) numbers refer to running words.

<table>
<thead>
<tr>
<th></th>
<th>Chinese</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>train:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>4.1M</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>78.3M</td>
<td>85.9M</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>382K</td>
<td>817K</td>
</tr>
<tr>
<td>Neural Network Vocabulary</td>
<td>169K</td>
<td>128K</td>
</tr>
<tr>
<td><strong>DEV12-tune (dev):</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1842</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>37.1K</td>
<td>46.3K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>5,993</td>
<td>5,573</td>
</tr>
<tr>
<td>OOVs</td>
<td>66 (0.2%)</td>
<td>78 (0.2%)</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>193 (0.5%)</td>
<td>152 (0.3%)</td>
</tr>
<tr>
<td><strong>DEV12-dev:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1844</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>37.7K</td>
<td>48.7K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>5,730</td>
<td>5,521</td>
</tr>
<tr>
<td>OOVs</td>
<td>52 (0.1%)</td>
<td>102 (0.2%)</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>184 (0.5%)</td>
<td>535 (1.1%)</td>
</tr>
<tr>
<td><strong>P1R6-dev:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1124</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>23.4K</td>
<td>29.1K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>4,259</td>
<td>3,700</td>
</tr>
<tr>
<td>OOVs</td>
<td>23 (0.1%)</td>
<td>80 (0.3%)</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>95 (0.4%)</td>
<td>367 (1.3%)</td>
</tr>
</tbody>
</table>
Table A.5: Corpus statistics for the discussion-forum portion of the Chinese→English BOLT training data. Out-of-vocabulary (OOV) numbers refer to running words. The neural network vocabulary is created by removing singleton words.

<table>
<thead>
<tr>
<th></th>
<th>Chinese</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>train (in-domain):</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>64.8K</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>1.4M</td>
<td>1.7M</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>40.2K</td>
<td>29.2K</td>
</tr>
<tr>
<td>Neural Network Vocabulary</td>
<td>27.3K</td>
<td>19.0K</td>
</tr>
<tr>
<td><strong>DEV12-tune (dev):</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1842</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>37.1K</td>
<td>46.3K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>5,993</td>
<td>5,573</td>
</tr>
<tr>
<td>OOVs</td>
<td>807 (2.1%)</td>
<td>545 (1.2%)</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>1,148 (3.1%)</td>
<td>841 (1.8%)</td>
</tr>
<tr>
<td><strong>DEV12-dev:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1844</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>37.7K</td>
<td>48.2K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>5,730</td>
<td>5,521</td>
</tr>
<tr>
<td>OOVs</td>
<td>536 (1.4%)</td>
<td>1,363 (2.8%)</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>844 (2.2%)</td>
<td>1,641 (3.4%)</td>
</tr>
<tr>
<td><strong>P1R6-dev:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1124</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>23.4K</td>
<td>29.1K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>4,259</td>
<td>3,700</td>
</tr>
<tr>
<td>OOVs</td>
<td>468 (2.0%)</td>
<td>800 (2.8%)</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>730 (3.1%)</td>
<td>979 (3.4%)</td>
</tr>
</tbody>
</table>
Table A.6: Corpus statistics for the discussion-forum portion of the Chinese→English BOLT training data used to train the recurrent neural language model. Out-of-vocabulary (OOV) numbers refer to running words. The neural network vocabulary is created by removing singleton words.

<table>
<thead>
<tr>
<th></th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>train (in-domain):</td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>354K</td>
</tr>
<tr>
<td>Running Words</td>
<td>8.6M</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>83.1K</td>
</tr>
<tr>
<td>Neural Network Vocabulary</td>
<td>53.3K</td>
</tr>
<tr>
<td>DEV12-tune (dev):</td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1842</td>
</tr>
<tr>
<td>Running Words</td>
<td>46.3K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>5,573</td>
</tr>
<tr>
<td>OOVs</td>
<td>258 (0.6%)</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>339 (0.7%)</td>
</tr>
<tr>
<td>DEV12-dev:</td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1844</td>
</tr>
<tr>
<td>Running Words</td>
<td>48.2K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>5,521</td>
</tr>
<tr>
<td>OOVs</td>
<td>1,104 (2.3%)</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>1,178 (2.4%)</td>
</tr>
<tr>
<td>P1R6-dev:</td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1124</td>
</tr>
<tr>
<td>Running Words</td>
<td>29.1K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>3,700</td>
</tr>
<tr>
<td>OOVs</td>
<td>629 (2.2%)</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>684 (2.4%)</td>
</tr>
</tbody>
</table>
Table A.7: Corpus statistics for the Arabic→English BOLT task. Out-of-vocabulary (OOV) numbers refer to running words.

<table>
<thead>
<tr>
<th></th>
<th>Arabic</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>train:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>922K</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>13.8M</td>
<td>15.5M</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>285K</td>
<td>203K</td>
</tr>
<tr>
<td>Neural Network Vocabulary</td>
<td>138.7K</td>
<td>87.4K</td>
</tr>
<tr>
<td><strong>DEV12-tune (dev):</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1,219</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>17.8K</td>
<td>21.1K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>4,294</td>
<td>3,038</td>
</tr>
<tr>
<td>OOVs</td>
<td>187 (1.1%)</td>
<td>60  (0.3%)</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>245 (1.4%)</td>
<td>79 (0.4%)</td>
</tr>
<tr>
<td><strong>DEV12-dev:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1,510</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>27.3K</td>
<td>32.2K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>6027</td>
<td>4000</td>
</tr>
<tr>
<td>OOVs</td>
<td>242 (0.9%)</td>
<td>143 (0.4%)</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>326 (1.2%)</td>
<td>173 (0.5%)</td>
</tr>
<tr>
<td><strong>P1R6-dev:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1,137</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>18.1K</td>
<td>21.6K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>3,917</td>
<td>2,780</td>
</tr>
<tr>
<td>OOVs</td>
<td>154 (0.9%)</td>
<td>51 (0.2%)</td>
</tr>
<tr>
<td>Neural Network OOVs</td>
<td>235 (1.3%)</td>
<td>88 (0.4%)</td>
</tr>
</tbody>
</table>
A.3 Conference on Machine Translation

The *Conference on Machine Translation* (WMT) is an annual evaluation campaign that provides training and test data for many European language pairs into and out of English. We use the news task of WMT which focuses on the translation of news text.

### A.3.1 WMT 2016 English→Romanian

Table A.8: Corpus statistics for the WMT 2016 English→Romanian task. Out-of-vocabulary (OOV) numbers refer to running words. The neural network vocabulary is the top most frequent 50k words, computed separately for each of the source and target corpora. We also use another vocabulary referred to as the large neural network vocabulary in the table. This vocabulary is selected using the top frequent 56k source and 81k target words.

<table>
<thead>
<tr>
<th></th>
<th>English</th>
<th>Romanian</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>train:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>604K</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>15.5M</td>
<td>15.8M</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>92.3K</td>
<td>128.4K</td>
</tr>
<tr>
<td>Neural Network Word Vocabulary</td>
<td>50K</td>
<td>50K</td>
</tr>
<tr>
<td>Neural Network Large Word Vocabulary</td>
<td>56K</td>
<td>81K</td>
</tr>
<tr>
<td><strong>newsdev2016_1 (dev):</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1,000</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>24.7K</td>
<td>26.7K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>5,217</td>
<td>6,407</td>
</tr>
<tr>
<td>OOVs</td>
<td>938 (3.8%)</td>
<td>1,270 (4.8%)</td>
</tr>
<tr>
<td>Neural Network Word OOVs</td>
<td>1,230 (5.0%)</td>
<td>2,051 (7.7%)</td>
</tr>
<tr>
<td>Neural Network Large Word OOVs</td>
<td>1,137 (4.6%)</td>
<td>1,553 (5.8%)</td>
</tr>
<tr>
<td><strong>newstest2016:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentences</td>
<td>1,999</td>
<td></td>
</tr>
<tr>
<td>Running Words</td>
<td>48.0K</td>
<td>49.8K</td>
</tr>
<tr>
<td>Vocabulary</td>
<td>7,375</td>
<td>10,301</td>
</tr>
<tr>
<td>OOVs</td>
<td>1,307 (2.8%)</td>
<td>2,539 (5.1%)</td>
</tr>
<tr>
<td>Neural Network Word OOVs</td>
<td>1,778 (3.7%)</td>
<td>4,260 (8.6%)</td>
</tr>
<tr>
<td>Neural Network Large Word OOVs</td>
<td>1,680 (3.5%)</td>
<td>3,217 (6.5%)</td>
</tr>
</tbody>
</table>

### A.3.2 WMT 2017 German→English

The corpus statistics for the training, development and test data of the German→English task provided for the *2017 Second Conference on Machine Translation* are presented in Table A.9. The data consists of common crawl, news commentary and the European parliament training sets. The data is filtered by removing sentences longer than 100 words. In addition, we remove sentences that have 5 or more consequent unaligned source words. The alignment used for filtering is the IBM4 alignment. Table A.10 shows the subword-level corpus statistics. This data includes the common crawl, news commentary, European parliament, and rapid data sets, without alignment filtering. We use newstest2015 as the development set and newstest2017 as the evaluation set. The data is filtered using alignment information by disregarding sentence pairs that have 5 or more consequent unaligned target or source words.

---

3http://www.statmt.org/wmt17
Table A.9: Corpus statistics for the bilingual data provided for the WMT 2017 German→English translation task. The data is filtered using heuristics to discard sentence pairs that have bad word alignment. Out-of-vocabulary (OOV) numbers refer to running words.

<table>
<thead>
<tr>
<th></th>
<th>German</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>train:</td>
<td>Sentences</td>
<td>3.6M</td>
</tr>
<tr>
<td></td>
<td>Running Words</td>
<td>84.5M</td>
</tr>
<tr>
<td></td>
<td>Vocabulary</td>
<td>671K</td>
</tr>
<tr>
<td></td>
<td>Neural Network Subword Vocabulary</td>
<td>50K</td>
</tr>
<tr>
<td></td>
<td>Neural Network Word Vocabulary</td>
<td>188K</td>
</tr>
<tr>
<td>newstest2015 (dev):</td>
<td>Sentences</td>
<td>2,169</td>
</tr>
<tr>
<td></td>
<td>Running Words</td>
<td>48.3K</td>
</tr>
<tr>
<td></td>
<td>Vocabulary</td>
<td>9,739</td>
</tr>
<tr>
<td></td>
<td>OOVs</td>
<td>507 (1.0%)</td>
</tr>
<tr>
<td></td>
<td>Neural Network Subword OOVs</td>
<td>0 (0.0%)</td>
</tr>
<tr>
<td></td>
<td>Neural Network Word OOVs</td>
<td>1,265 (2.6%)</td>
</tr>
<tr>
<td>newstest2017:</td>
<td>Sentences</td>
<td>3,004</td>
</tr>
<tr>
<td></td>
<td>Running Words</td>
<td>67.2K</td>
</tr>
<tr>
<td></td>
<td>Vocabulary</td>
<td>12,382</td>
</tr>
<tr>
<td></td>
<td>OOVs</td>
<td>741 (1.1%)</td>
</tr>
<tr>
<td></td>
<td>Neural Network Subword OOVs</td>
<td>0 (0.0%)</td>
</tr>
<tr>
<td></td>
<td>Neural Network Word OOVs</td>
<td>1,923 (2.9%)</td>
</tr>
</tbody>
</table>

Table A.10: Subword-level corpus statistics for the bilingual data provided for the WMT 2017 German→English translation task. The subword units are BPEs generated using 50K merge operations. Out-of-vocabulary (OOV) numbers refer to running subwords.

<table>
<thead>
<tr>
<th></th>
<th>German</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>train:</td>
<td>Sentences</td>
<td>5.9M</td>
</tr>
<tr>
<td></td>
<td>Running Subwords</td>
<td>137M</td>
</tr>
<tr>
<td></td>
<td>Subword Vocabulary</td>
<td>50K</td>
</tr>
<tr>
<td>newstest2015 (dev):</td>
<td>Sentences</td>
<td>2,169</td>
</tr>
<tr>
<td></td>
<td>Running Subwords</td>
<td>55.0K</td>
</tr>
<tr>
<td></td>
<td>Subword Vocabulary</td>
<td>10,636</td>
</tr>
<tr>
<td></td>
<td>OOVs</td>
<td>5 (0.0%)</td>
</tr>
<tr>
<td>newstest2017:</td>
<td>Sentences</td>
<td>3,004</td>
</tr>
<tr>
<td></td>
<td>Running Subwords</td>
<td>76.7K</td>
</tr>
<tr>
<td></td>
<td>Subword Vocabulary</td>
<td>9,652</td>
</tr>
<tr>
<td></td>
<td>OOVs</td>
<td>12 (0.0%)</td>
</tr>
</tbody>
</table>
**List of Figures**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Illustration of word alignment</td>
<td>9</td>
</tr>
<tr>
<td>4.1</td>
<td>A class-factored output layer</td>
<td>20</td>
</tr>
<tr>
<td>4.2</td>
<td>The feedforward lexicon model</td>
<td>21</td>
</tr>
<tr>
<td>4.3</td>
<td>The feedforward alignment model</td>
<td>23</td>
</tr>
<tr>
<td>4.4</td>
<td>LM using the recurrent neural network architecture</td>
<td>24</td>
</tr>
<tr>
<td>4.5</td>
<td>Unfolded recurrent neural network language model</td>
<td>24</td>
</tr>
<tr>
<td>4.6</td>
<td>Recurrent neural network translation model with and without target input</td>
<td>25</td>
</tr>
<tr>
<td>4.7</td>
<td>Example of the bidirectional lexical model</td>
<td>27</td>
</tr>
<tr>
<td>4.8</td>
<td>Example of the bidirectional lexical model with source-target state pairing</td>
<td>29</td>
</tr>
<tr>
<td>4.9</td>
<td>Example of the bidirectional alignment model</td>
<td>31</td>
</tr>
<tr>
<td>4.10</td>
<td>Example of the attention-based recurrent model</td>
<td>33</td>
</tr>
<tr>
<td>4.11</td>
<td>Recurrent dependencies of the RNN attention model</td>
<td>35</td>
</tr>
<tr>
<td>4.12</td>
<td>Source-to-target multi-head attention component</td>
<td>37</td>
</tr>
<tr>
<td>4.13</td>
<td>Dependencies in self-attentive layers</td>
<td>38</td>
</tr>
<tr>
<td>4.14</td>
<td>The transformer model architecture</td>
<td>40</td>
</tr>
<tr>
<td>4.15</td>
<td>Attention map for transformer and alignment-assisted transformer systems</td>
<td>41</td>
</tr>
<tr>
<td>4.16</td>
<td>Recurrent dependencies of the RNN attention model with alignment bias</td>
<td>42</td>
</tr>
<tr>
<td>4.17</td>
<td>Dependencies in alignment-assisted transformer</td>
<td>43</td>
</tr>
<tr>
<td>4.18</td>
<td>Source-to-target multi-head attention with alignment augmentation</td>
<td>44</td>
</tr>
<tr>
<td>5.1</td>
<td>Alignment conversion example</td>
<td>56</td>
</tr>
<tr>
<td>5.2</td>
<td>Forced-alignment flowchart</td>
<td>58</td>
</tr>
<tr>
<td>5.3</td>
<td>Search graph example of alignment-based neural machine translation</td>
<td>61</td>
</tr>
<tr>
<td>5.4</td>
<td>Decoding speed and translation quality with a class-factored output layer</td>
<td>74</td>
</tr>
<tr>
<td>5.5</td>
<td>Alignment-based log-linear weight tuning</td>
<td>75</td>
</tr>
<tr>
<td>5.6</td>
<td>Alignment pruning translation time and quality for batch size 1</td>
<td>76</td>
</tr>
<tr>
<td>5.7</td>
<td>Alignment pruning translation time and quality for batch size 5</td>
<td>76</td>
</tr>
<tr>
<td>6.1</td>
<td>Example of word-based translation</td>
<td>90</td>
</tr>
<tr>
<td>6.2</td>
<td>Example of phrase-based translation</td>
<td>90</td>
</tr>
<tr>
<td>6.3</td>
<td>Illustration of phrase segmentation</td>
<td>92</td>
</tr>
<tr>
<td>6.4</td>
<td>Excerpt of a search graph</td>
<td>96</td>
</tr>
<tr>
<td>6.5</td>
<td>Illustration of word alignment modification to resolve multiple and unaligned words</td>
<td>103</td>
</tr>
<tr>
<td>6.6</td>
<td>Approximation error due to target history truncation</td>
<td>109</td>
</tr>
<tr>
<td>6.7</td>
<td>Decoding vs. N-best rescoring using a recurrent language model</td>
<td>110</td>
</tr>
<tr>
<td>6.8</td>
<td>Cache hit ratios</td>
<td>111</td>
</tr>
</tbody>
</table>
### List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Architecture comparison of lexical models</td>
<td>54</td>
</tr>
<tr>
<td>5.2</td>
<td>Architecture comparison of alignment models</td>
<td>54</td>
</tr>
<tr>
<td>5.3</td>
<td>Context comparison for lexical models</td>
<td>55</td>
</tr>
<tr>
<td>5.4</td>
<td>Context comparison for alignment models</td>
<td>55</td>
</tr>
<tr>
<td>5.5</td>
<td>Comparison of standard and alignment-based neural machine translation systems.</td>
<td>65</td>
</tr>
<tr>
<td>5.6</td>
<td>WMT 2016 English→Romanian results for baseline and alignment-based systems.</td>
<td>67</td>
</tr>
<tr>
<td>5.7</td>
<td>BOLT Chinese→English results for baseline and alignment-based systems.</td>
<td>68</td>
</tr>
<tr>
<td>5.8</td>
<td>WMT 2017 German→English results for baseline and alignment-based systems.</td>
<td>69</td>
</tr>
<tr>
<td>5.9</td>
<td>Frequent words vs byte-pair encoding vocabulary</td>
<td>69</td>
</tr>
<tr>
<td>5.10</td>
<td>A comparison between alignment-based feedforward and recurrent systems WMT English→Romanian.</td>
<td>71</td>
</tr>
<tr>
<td>5.11</td>
<td>A comparison between alignment-based feedforward and recurrent systems on BOLT Chinese→English.</td>
<td>71</td>
</tr>
<tr>
<td>5.12</td>
<td>A comparison between alignment-based feedforward and recurrent systems on WMT German→English.</td>
<td>72</td>
</tr>
<tr>
<td>5.13</td>
<td>BOLT Chinese→English results highlighting the effect of feedforward and recurrent lexical and alignment models.</td>
<td>72</td>
</tr>
<tr>
<td>5.14</td>
<td>WMT English→Romanian results using variants of the bidirectional RNN lexical model.</td>
<td>73</td>
</tr>
<tr>
<td>5.15</td>
<td>Effect of alignment block out in training and decoding.</td>
<td>74</td>
</tr>
<tr>
<td>5.16</td>
<td>Alignment variants: time vs. translation quality.</td>
<td>76</td>
</tr>
<tr>
<td>5.17</td>
<td>Alignment error rate results for the German→English and Chinese→English systems.</td>
<td>78</td>
</tr>
<tr>
<td>5.18</td>
<td>Dictionary statistics.</td>
<td>79</td>
</tr>
<tr>
<td>5.19</td>
<td>Dictionary experiments.</td>
<td>79</td>
</tr>
<tr>
<td>5.20</td>
<td>Hard vs. soft alignment extraction.</td>
<td>81</td>
</tr>
<tr>
<td>5.21</td>
<td>WMT 2016 English→Romanian forced-alignment training results.</td>
<td>83</td>
</tr>
<tr>
<td>5.22</td>
<td>BOLT Chinese→English forced-alignment training results.</td>
<td>83</td>
</tr>
<tr>
<td>5.23</td>
<td>WMT 2017 German→English forced-alignment training results.</td>
<td>83</td>
</tr>
<tr>
<td>5.24</td>
<td>Source coverage statistics of the training data for WMT 2016 English→Romanian.</td>
<td>83</td>
</tr>
<tr>
<td>5.25</td>
<td>Source coverage statistics of the training data for BOLT Chinese→English.</td>
<td>83</td>
</tr>
<tr>
<td>5.26</td>
<td>Source coverage statistics of the training data for WMT 2017 German→English.</td>
<td>84</td>
</tr>
<tr>
<td>5.27</td>
<td>Sample translations highlighting positive examples from the alignment-assisted transformer system compared to the transformer baseline.</td>
<td>85</td>
</tr>
<tr>
<td>5.28</td>
<td>Sample translations highlighting negative examples from the alignment-assisted transformer system compared to the transformer baseline.</td>
<td>86</td>
</tr>
<tr>
<td>6.1</td>
<td>RNN decoding vs. rescoring results on IWSLT German→English.</td>
<td>106</td>
</tr>
</tbody>
</table>
List of Tables

6.2 RNN decoding vs. rescoring results on BOLT Arabic→English. .................. 108
6.3 RNN decoding vs. rescoring results on BOLT Chinese→English. ................. 108
6.4 Recurrent language model approximation effect in decoding vs. rescoring. .... 109
6.5 Effect of caching order on translation quality. ........................................ 110
6.6 Effect of caching on translation speed. .................................................. 112

A.1 IWSLT 2013 German→English in-domain data statistics. ......................... 117
A.2 IWSLT 2013 German→English data statistics for the recurrent language model. . 118
A.3 BOLT Chinese→English data statistics. ............................................... 119
A.4 BOLT Chinese→English data statistics. ............................................... 119
A.5 BOLT Chinese→English discussion forum data statistics. ......................... 120
A.6 BOLT Chinese→English discussion forum data statistics. ......................... 121
A.7 BOLT Arabic→English data statistics. ............................................... 122
A.8 WMT English→Romanian data statistics. ............................................ 123
A.9 WMT 2017 German→English data statistics. ........................................ 124
A.10 WMT 2017 German→English subword data statistics. .......................... 124


BIBLIOGRAPHY


