Bachelorarbeit im Fach Informatik
Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik 6
Prof. Dr.-Ing. H. Ney

Alignment Methods for Attention-based Neural Machine Translation

26. September 2016

vorgelegt von:
Arne Fabian Nix
Matrikelnummer 331423

Gutachter:
Prof. Dr.-Ing. H. Ney
Prof. G. Lakemeyer, Ph. D.

Betreuer:
Dipl. Inform. J. Peter

Erklärung

Nix, Arne Fabian — 331423

Name, Vorname — Matrikelnummer (freiwillige Angabe)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende ~~Arbeit~~/Bachelorarbeit/ ~~Masterarbeit*~~ mit dem Titel

Alignment Methods for Attention-based Neural Machine Translation

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Aachen, 26. September 2016 — _____

Ort, Datum — Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Aachen, 26. September 2016 — _____

Ort. Datum — Unterschrift

Abstract

The attention-based neural machine translation (NMT) approach has outperformed traditional phrase based approaches on numerous translation tasks in recent evaluations. Attention-based NMT uses an end-to-end trainable artificial neural network (ANN) to produce the translation for a given sentence. As a part of the translation, the ANN implicitly produces a weighting over the words of the source sentence, which can be interpreted as a soft alignment from source to target sentence.

In this bachelor thesis, the attention mechanism and the resulting alignments are examined. An evaluation is conducted that shows a clear correlation between the alignment score produced by traditional and recently proposed alignment evaluation methods with the translation quality. Moreover, in a comparison of NMT alignments with state-of-the-art statistical alignments, we can see that the soft alignments the NMT approach produces offer room for improvement. Also, the experiments indicated that the attention-based alignments have a significant impact on the translation quality. Hence, we propose multiple approaches to extend and improve the attention mechanism and thereby the alignment as well as the translation quality. Among these are two methods that resulted in an improvement of more than one BLEU %.

Additionally, we also modified the attention-based NMT model to generate alignments for given source and target sentence pairs. With this model, we were able to halve the AER and SAER scores, compared to the standard NMT attention model, and outperform state-of-the-art alignments produced by traditional statistical alignment models.

Contents

Chapter 1

Introduction

Machine translation (MT) describes the task to automate the translation of a given sentence from one language to another. Usually, MT can be seen as a decision process, which means that the user depends on the system to automatically decide on a good translation for him. This leads us to the main problem MT tries to solve:

Definition 1 (Translation Modeling Problem)
Given a training set $\mathcal{T}$ *of bilingual sentence pairs* $(\tilde{f}_1^{\tilde{J}}, \tilde{e}_1^{\tilde{I}})$ *one seeks to minimize decision errors in translation of a separate set* $\mathcal{V}$ *of unseen test data* $(f_1^J, \hat{e}_1^{\hat{I}})$*, i.e. to maximize* $Pr(\hat{e}_1^{\hat{I}}|f_1^J)$ *for all* $(f_1^J, \hat{e}_1^{\hat{I}}) \in \mathcal{V}$*.*

This problem is hard, mainly due to the complexity of human languages. Every language is different and especially ambiguities in translation may be intuitive to humans, since they understand the context, but make it hard to automate the translation process for MT.

Nevertheless, there have been several different approaches that tried to solve this problem. This thesis will focus on *statistical machine translation (SMT)*, which means that we use statistical methods to model the translation process. For these statistical models, we will apply a method humans do intuitively without noticing it. Humans draw connections between the words in the given sentence (*source sentence*) and the words in its translation (*target sentence*). The set of such connections that covers the whole source sentence forms an *alignment*. In the process of SMT, we include an alignment model which will compute an alignment as an important intermediate step of translation.

Definition 2 (Alignment Problem)
Given a pair (f_1^J, e_1^I) *of source and target sentence, the goal is to find a mapping* $j \to i = a_j$ *for all* $j = 1, \ldots, J$ *that leads to a minimization of decision errors for the translation of* (f_1^J, e_1^I)*, i.e. to maximize* $Pr(e_1^I|f_1^J, a_1^J)$*.*[1]

[1] In chapter 2, we will see a different inclusion of the alignment since we model $Pr(f_1^J|e_1^I)$ there.

In the last decade, there have been many great successes in different fields of machine learning, which led to a renaissance of neural networks.

The same has happened in the field of SMT, where neural networks are used to replace the traditional methods, by directly modeling the translation probability. Methods that follow this approach are summarized under the term *neural machine translation (NMT)*. In 2014, the first method that showed results competitive to traditional SMT was presented. This approach was extended to the *attention-based* approach that outperformed traditional SMT on many tasks in the last two years. The attention-based approach includes, similar to traditional SMT, an implicit alignment step (the *attention mechanism*) that seems to be crucial for a good translation. The great success of attention-based NMT drew much attention to this particular approach and resulted in numerous publications on this topic. Even though many of these publications tried to improve the translation by using more advanced attention methods, the actual influence that the attention mechanism has on the translation quality has been subject to little research.

Therefore the goal of this thesis is first to find a suitable evaluation criterion to measure the quality of the alignment produced by the attention mechanism and the resulting translation quality. We investigate existing attention methods and compare the resulting alignment with traditional alignment methods. Finally, we introduce several modifications to the standard attention mechanism to add more context information. We show in experiments on two translation tasks that these methods outperform the standard NMT approach in translation and alignment quality.

1.1 Related Work

Statistical machine translation, which has been the state-of-the-art for many years and which will serve as a comparison for the alignment methods throughout this thesis, has been introduced by Brown et al. [Brown & Pietra$^+$ 93] in 1993 and was extended to the phrase-based approach by Och et al. [Och & Ney 04] in 2004. Also worth mentioning here is the HMM alignment method as it was introduced by Vogel et al. [Vogel & Ney$^+$ 96] since it extends the alignment methods by adding the first-order dependency.

Sutskever et al. [Sutskever & Vinyals$^+$ 14] and Cho et al. [Cho & van Merrienboer$^+$ 14] both introduced the encoder-decoder approach which was the first example of an NMT approach that is competitive to phrase-based MT. The encoder-decoder approach was extended to the attention-based approach [Bahdanau & Cho$^+$ 15] which today represents the state-of-the-art model for NMT since it outperforms phrase-based MT systems on many tasks. Therefore, as the title suggests, all methods presented in this thesis will be based on the attention-based NMT approach.

When it comes to statistical alignments, there have been many evaluations of the alignment quality [Och & Ney 03, Ayan & Dorr 06]. Moreover, the alignment evaluation method itself as well as the question what influence the alignment quality has on the resulting translation [Fraser & Marcu 07, Ayan & Dorr 06, Vilar & Popovic$^+$ 06] have also been subject to investigations.

Attention-based NMT also produces an alignment as an intermediate step of the translation process, which is widely believed to have a great impact on translation quality. But, to our knowledge, there have only been three empirical investigations of the alignment quality [Tu & Lu$^+$ 16,Mi & Wang$^+$ 16,Sankaran & Mi$^+$ 16] to date. These three methods use SAER and AER [Tu & Lu$^+$ 16], respectively F1 scores [Mi & Wang$^+$ 16, Sankaran & Mi$^+$ 16], to compare the alignment quality of different attention models. Nevertheless, we do not know whether these alignment evaluation methods are even suitable for soft alignments as the attention mechanism generates them. Since attention-based NMT has proven to be competitive to phrase-based NMT, there have been several different approaches to improving the translation quality by modifying the attention mechanism. Most methods simply add different features to the attention computation itself [Tu & Lu$^+$ 16, Mi & Sankaran$^+$ 16, Sankaran & Mi$^+$ 16,Feng & Liu$^+$ 16,Cohn & Hoang$^+$ 16] and there are also attempts to change the attention mechanism itself [Zhang & Xiong$^+$ 16]. Other methods try to improve by extending the network error that is fed back to the attention mechanism for training and adds new information in training [Chen & Matusov$^+$ 16, Mi & Wang$^+$ 16].

1.2 Outline

In this thesis, we discuss MT and the attention-based approach proposed by Bahdanau et al. [Bahdanau & Cho$^+$ 15]. We start with a brief introduction to traditional SMT models in chapter 2 where we cover both the word-based approach (section 2.1) as well as the more state-of-the-art phrase-based approach (section 2.4). In that chapter, we will also go into some detail about traditional statistical alignment methods and GIZA++ (section 2.2), since they still represent the state-of-the-art for statistical alignments.

Chapter 3 gives a broad overview of artificial neural networks, the training methods (section 3.4) and the special architectures of convolutional (section 3.2) and recurrent neural networks (section 3.3) in order to cover all the basic techniques that get utilized in the later parts of this thesis.

Afterwards, in chapter 4, we show how these techniques get applied in NMT. There we discuss the encoder-decoder approach (section 4.1) as well as the attention-based approach (section 4.2) as they are the most prominent methods in NMT.

In the chapter "Advanced attention methods" (chapter 5) we explain different state-of-the-art methods to improve the attention (section 5.1) and introduce the approaches we are investigating in this thesis. Among these are methods to feed back alignment information (section 5.2) and to change the attention computation itself so that it preserves more context information (section 5.3). Furthermore, we are extending the loss function to provide a better error signal to the attention mechanism (section 5.4) and we modify the attention computation to artificially improve the alignment by introducing knowledge of the target sentence (section 5.5).

Finally, we conclude this thesis by performing experiments to determine the translation and alignment quality of all attention methods we presented (chapter 6). To further investigate the impact these methods have on the alignment itself, we compared different alignment evaluation measures (section 6.3) and noticed that all these methods highly correlate with translation quality. With this in mind, we evaluated the alignment quality of the methods we introduced (section 6.4) and used the method we called "Alignment Foresight" to generate an alignment that beats GIZA-alignments in alignment quality (section 6.3.3).

Chapter 2

Statistical Machine Translation

As described in the previous chapter, the general goal of machine translation is to find the translation hypothesis $\hat{e}_1^{\hat{I}}$, that represents the optimal translation of a given sentence f_1^J in the source language, i.e. to find a mapping:

$$f_1^J \rightarrow (\hat{I}, \hat{e}_1^{\hat{I}})(f_1^J) = \underset{I, e_1^I}{\operatorname{argmax}} \{ Pr(e_1^I | f_1^J) \} \tag{2.1}$$

The first widely successful approach to find such a mapping was the *word-based approach* that was the basis for the *phrase-based approach*, which has been the state-of-the-art method for many years. Both of these methods will be presented briefly in the upcoming sections of this chapter.

2.1 Word-based Approach

For the word based approach, introduced by Brown, Della Pietra, Della Pietra and Mercer in 1993 [Brown & Pietra$^+$ 93], we reformulate the maximization from equation (2.1). Therefore, we split the maximization into language model $Pr(e_1^I)$ and translation model $Pr(f_1^J | e_1^I)$, by applying *Bayes theorem* and dropping components that are unrelated to the maximization.

$$\underset{I, e_1^I}{\operatorname{argmax}} \{ Pr(e_1^I | f_1^J) \} = \underset{I, e_1^I}{\operatorname{argmax}} \left\{ \frac{Pr(e_1^I, f_1^J)}{Pr(f_1^J)} \right\} \tag{2.2}$$

$$= \underset{I, e_1^I}{\operatorname{argmax}} \{ Pr(e_1^I) \cdot Pr(f_1^J | e_1^I) \} \tag{2.3}$$

The advantage here is that those two probability distributions are independent from each other and therefore it is also possible to model them separately on, possibly different and orders of magnitude more, data.

Brown et al. introduced an even further separation by introducing hidden alignment variables a_1^J for the translation model

$$Pr(f_1^J | e_1^I) = \sum_{a_1^J} Pr(f_1^J, a_1^J | e_1^I) \tag{2.4}$$

and splitting it into three separate models.

$$Pr(f_1^J|e_1^I) = \sum_{a_1^J} Pr(J|e_1^I) \cdot Pr(f_1^J, a_1^J|J, e_1^I) \tag{2.5}$$

$$= \sum_{a_1^J} Pr(J|e_1^I) \cdot Pr(a_1^J|J, e_1^I) \cdot Pr(f_1^J|a_1^J, J, e_1^I) \tag{2.6}$$

This way, the translation model is a product of length probability $Pr(J|e_1^I)$, alignment probability $Pr(a_1^J|J, e_1^I)$ and lexicon probability $Pr(f_1^J|a_1^J, J, e_1^I)$.

In the following, we will present the models that were introduced by Brown et al. as part of the IBM-1 and IBM-2 model for the length and lexicon distribution. In the next section, we will go into further detail about language model and in particular the alignment model, as this is most important for the rest of the thesis.

2.1.1 Length Model

For the length model, Brown et al. apply a strong model assumption, which states that the length J of the source sentence is only dependent on the length of the target sentence I. This way we get

$$Pr(J|e_1^I) = p(J|I) \tag{2.7}$$

as the length model. Due to the low dimensionality of this length model ($I \times J$), it is possible to model it with a *Poisson distribution* with parameter λ_I or use a full table as an unconstrained distribution.

2.1.2 Lexicon Model

To derive the lexicon model introduced by Brown et al., we first need to use the chain rule:

$$Pr(f_1^J|a_1^J, J, e_1^I) = \prod_{j=1}^{J} Pr(f_j|f_1^{j-1}, a_1^J, J, e_1^I) \tag{2.8}$$

We further apply a strong model assumption by dropping all dependencies except for the corresponding target word specified by the alignment a_j.

$$Pr(f_1^J|a_1^J, J, e_1^I) = \prod_{j=1}^{J} p(f_j|e_{a_j}) \tag{2.9}$$

2.2 Alignment Model

An alignment, as defined by Brown et al. [Brown & Pietra$^+$ 93], is an object that indicates the corresponding words in a parallel text. Therefore, the alignment of two sentences f_1^J and e_1^I can also be defined, following Och and Ney [Och & Ney 03], as a mapping from source to target sentence positions:

$$j \rightarrow i = a_j \text{ for all } j = 1, \ldots, J \tag{2.10}$$

This results in the restrictions that the alignment a_1^J assigns each source word to exactly one target word. If a source word can not be aligned with any target word, it will be aligned to an empty word e_0, which is added to the target sentence. As described before, we introduce the alignment as a hidden variable and therefore obtain a separate alignment probability $Pr(a_1^J|J, e_1^I)$ as one part of the translation model. Brown et al. [Brown & Pietra$^+$ 93] (IBM-1 to IBM-5) as well as Vogel et al. [Vogel & Ney$^+$ 96] (HMM) introduced different, increasingly sophisticated models, that can also be used in a hierarchical training process where a simpler model serves as initialization for the training of the next model. The specifics of each model will now be presented briefly. A more detailed explanation of all the models can be found in the paper by Och and Ney [Och & Ney 03].

IBM-1 Alignment Model

For IBM-1, the alignment model $p(i = a_j|a_1^J, J, e_1^I)$ is defined as the uniform distribution $\frac{1}{(I+1)}$. This introduces a very strict model assumption, by completely ignoring the word order:

$$Pr(a_1^J|J, e_1^I) = \prod_{j=1}^{J} p(i = a_j|a_1^J, J, e_1^I) = \frac{1}{(I+1)^J} \tag{2.11}$$

IBM-2 Alignment Model

Adding the dependency on the source position j but still keeping the zero-order model assumption, we obtain the following model for IBM-2:

$$Pr(a_1^J|J, e_1^I) = \prod_{j=1}^{J} p(i = a_j|a_1^J, J, e_1^I) = \prod_{j=1}^{J} p(a_j|j, J, I) \tag{2.12}$$

It is also possible to ignore the dependency on J in order to reduce the number of parameters, such that we get:

$$p(a_j|j, J, I) = p(a_j|j, I) \tag{2.13}$$

HMM Alignment Model

Proven successful in speech recognition, Vogel et al. [Vogel & Ney$^+$ 96] transferred the HMM alignment context to alignments for statistical machine translation. Therefore, they introduced a first-order dependency:

$$Pr(a_1^J|J, e_1^I) = \prod_{j=1}^{J} p(i = a_j|a_1^J, J, e_1^I) = \prod_{j=1}^{J} p(a_j|a_{j-1}, I) \tag{2.14}$$

A further assumption that the alignment probabilities $p(i|i', I)$ depend only on the jump width $(i - i')$, results in the following model:

$$p(a_j = i|a_{j-1} = i', I) = \frac{c(i - i')}{\sum_{i''=1}^{I} c(i'' - i')} \tag{2.15}$$

where $\{c(i - i')\}$ is a set of non-negative parameters.

2.2.1 Fertility-Based Alignment Models

A common problem that the methods described above have to face is the so-called *coverage problem*. This describes the condition that a model might fall into one of two extreme cases for the translation of certain sentences:

- Over-translation: The system unnecessarily translates some words multiple times
- Under-translation: The system mistakenly leaves some word untranslated

The models IBM-3 to IBM-5 aim to prevent this behavior. For these more advanced models, Brown et al. [Brown & Pietra$^+$ 93] use an inverted mapping from target to source positions. Since each target word can be aligned to multiple source words, the inverted variant of mapping (2.10) defines a set of aligned source word for each target position $i = 0, \ldots, I$:

$$B : i \to B_i \subset \{1, \ldots, j, \ldots, J\} \tag{2.16}$$

where B_0 is the position of all source words that are aligned to the empty word. Utilizing this inverted alignment, we can rewrite equation (2.5) to obtain the inverted case of the translation model decomposition:

$$Pr(f_1^J, a_1^J|e_1^I) = Pr(f_1^J, B_0^I|e_1^I) \tag{2.17}$$

$$= Pr(B_0|B_1^I) \cdot \prod_{i=1}^{I} Pr(B_i|B_1^{i-1}, e_1^I) \cdot Pr(f_1^J|B_0^I, e_1^I) \tag{2.18}$$

$$= p(B_0|B_1^I) \cdot \prod_{i=1}^{I} p(B_i|B_{i-1}, e_i) \cdot \prod_{i=0}^{I} \prod_{j \in B_i} p(f_j|e_i) \tag{2.19}$$

Given the inverted alignment, we can easily define the *fertility*, which represents the number of source words aligned to a specific target position:

$$\phi_i = \sum_j \delta(a_j, i) = |B_i| \tag{2.20}$$

We can specify the model $p(B_0|B_1^I)$ of the alignment to the empty word as a uniform distribution using the corresponding fertility ϕ_0:

$$p(B_0|B_1^I) = \binom{J-\phi_0}{\phi_0}(1-p_1)^{J-2\phi_0}p_1^{\phi_0} \cdot \frac{1}{\phi_0!} \tag{2.21}$$

For IBM models 3 and 4, we replace $\phi_0!$ by J^{ϕ_0}, such that the alignment models for both empty and nonempty words are *deficient*, i.e. that the probabilities of all valid alignments sum to unity.

IBM-3 Alignment Model

Similar to the IBM-2 model, the IBM-3 model completely ignores the previous alignment B_{i-1} and therefore applies a zero-order dependency for the inverted case:

$$p(B_i|B_{i-1}, e_i) = p(\phi_i|e_i) \cdot \phi_i! \cdot \prod_{j \in B_i} p(j|i, J) \tag{2.22}$$

IBM-4 Alignment Model

We extend the inverse alignment model to a first-order model, by adding a dependency on the source word that was aligned before, or to be precise a dependency on the jump width from B_{ik-1} to B_{ik}:

$$p(B_i|B_{i-1}, e_i) = p(\phi_i|e_i) \cdot p_{=1}(B_{i1} - \overline{B_{\rho(i)}}|\ldots) \prod_{k=2}^{\phi_i} p_{>1}(B_{ik} - B_{ik-1}|\ldots) \tag{2.23}$$

Where $p_{=1}(\Delta j|\ldots)$ and $p_{>1}(\Delta j|\ldots)$ are first order alignment models, $\rho(i)$ gives the largest value $i' < i$ for which $|B_{i'}| > 0$. $\overline{B_{\rho(i)}}$ is the average of all elementes in $B_{\rho(i)}$.

IBM-5 Alignment Model

Finally, the IBM-5 model is a reformulation of model 4 that is supposed to avoid deficiency. We will not specify the formulas here, but instead, we refer to Brown et al. [Brown & Pietra$^+$ 93] for additional details on IBM-5 or any of the IBM models.

2.2.2 GIZA++

To enable the generation of suitable alignments with reasonable effort using the models described above, the *GIZA++* training software [Och & Ney 03] has been developed. This software implements all the mentioned alignment methods including some additional refinements and offers an interface to easily execute a hierarchical training process as mentioned in the introduction to this chapter.

Additional to some refinements to the alignment methods themselves, GIZA++ swaps source and target language to achieve a symmetrized alignment from source to target and from target to source. The two alignment sets A_1 and A_2 that result from this process can be combined using simple intersection $A_1 \cap A_2$ or union $A_1 \cup A_2$. GIZA++ also implements more advanced methods to combine the two alignments, like e.g. the following heuristic:

Starting from the intersection $A = A_1 \cap A_2$, alignment points (i, j) that occur in exclusively in one of the two alignments are added, if

- $(j, \cdot)$ and $(\cdot, i)$ do not occur in A

or

- the alignment point (i, j) has a horizontal $(i-1, j)$, $(i+1, j)$ or vertical $(i, j-1)$, $(i, j+1)$ neighbour in A
- and $A \cup \{(i, j)\}$ does not result in an alignment where one point has both horizontal and vertical neighbours.

Applying this heuristic or any simpler combination method results in a suitable alignment, if both alignments are reliable. To make this more likely, we need to find the optimal alignment for our models.

Viterbi Alignment

In most applications, we do not need all possible alignments but only the best hypothesized alignment, the *Viterbi alignment*. For the IBM-2 model and similarly for the IBM-1 model we can just perform an independent maximization for each position $j = 1, \ldots, J$, with complexity $\mathcal{O}(I \cdot J)$, due to the zero-order dependencies:

$$\hat{a}_1^J = \arg\max_{a_1^J} Pr(f_1^J, a_1^J | e_1^I) \tag{2.24}$$

$$= \arg\max_{a_1^J} \left\{ p(J|I) \cdot \prod_{j=1}^{J} [p(a_j|j, I) \cdot p(f_j|e_{a_j})] \right\} \tag{2.25}$$

$$= \left[\arg\max_{a_j} \{ p(a_j|j, I) \cdot p(f_j|e_{a_j}) \} \right]_{j=1}^{J} \tag{2.26}$$

The maximization of the HMM alignment can not be performed in the same way, but instead needs to be optimized using *dynamic programming*, which leads to a complexity of $\mathcal{O}(I^2 \cdot J)$. For the refined, fertility-based, alignment models such an efficient optimization is not possible. As suggested by Brown et al., one would instead do an iterative optimization w.r.t. the alignment probability starting with an Viterbi alignment that was estimated for a simpler model, e.g. an HMM model.

2.3 Language Model

A crucial part of a good translation is the language model $Pr(e_1^I)$, which should, in theory, be responsible for the generation of syntactically and semantically correct sentences in the target language.

Traditionally $Pr(e_1^I)$ would be modeled by applying the chain rule and a Markov assumption, which limits the model to a local context, i.e. a so-called *n-gram* dependency:

$$Pr(e_1^I) = \prod_{i=1}^{I} p(e_i|e_0^{i-1}) = \prod_{i=1}^{I} p(e_i|e_{i-n+1}^{i-1}) \tag{2.27}$$

with the special sentence start token e_0 that is introduced to represent the empty context for $p(e_1|e_0)$ at the beginning of the sentence. The conditional probability $p(e_i|e_{i-n+1}^{i-1})$ would often be estimated by relative frequencies with leaving one out [Ney & Essen$^+$ 95], i.e. *count-based* modeling.

Another possibility would be to use a multilayer perceptron (section 3.1) to model such probabilities with limited context.

Most recently, it was shown that recurrent neural networks (section 3.3) and especially long short-term memory networks (section 3.3.1) have the capability to compress and later uncompress context of arbitrary length into a fixed-size vector. Exploiting this, it is possible to compress the full context e_0^{m-1} into a fixed-size representation and use this to model e_m. Therefore we can drop the constraint on an n-gram dependency and obtain

$$Pr(e_1^I) = \prod_{i=1}^{I} p(e_i|e_0^{i-1}) \tag{2.28}$$

for language modeling with recurrent neural networks. However, if we use a straightforward implementation of such a language model, we would end up with an output layer that has the size of the vocabulary, which would make the corresponding

computations very costly. One commonly used way to speed this up would be a factorization of the output layer using word classes [Goodman 01]:

$$Pr(e_1^I) = \prod_{i=1}^{I} p(e_i|e_0^{i-1}) = \prod_{i=1}^{I} p(G(e_i)|e_0^{i-1}) \cdot p(e_i|G(e_i), e_0^{i-1}) \tag{2.29}$$

Since it is much easier to collect monolingual data, compared to bilingual data for translation, the language model is usually trained on much bigger data sets than the translation model. This is one advantage compared to models, like the neural machine translation model (chapter 4), which have no explicit separation of language and translation model.

2.4 Phrase-based Approach

The models we explained up to this point were all based on single words, but based on intuition and empirical evidence, we know that context is crucial for translation. One possibility to handle context in translation would be to translate phrases [Och & Ney 04], i.e. an arbitrary word group, instead of single words. Therefore, it is necessary first to produce an alignment of the bilingual sentence pairs using the methods described in section 2.2. With this, we can extract bilingual phrases $(\tilde{f}, \tilde{e})$, for which the following prerequisites have to be fulfilled:

- $\tilde{f}$ and $\tilde{e}$ are contiguous
- All words in $\tilde{f}$ are aligned only to words in $\tilde{e}$
- All words in $\tilde{e}$ are aligned only to words in $\tilde{f}$

The phrase extraction results in a segmentation into bilingual phrases:

$$\tilde{e}_k := e_{i_{k-1}+1}, \ldots, e_{i_k} \quad \text{for } k = 1, \ldots, K \tag{2.30}$$

$$\tilde{f}_k := f_{b_{k-1}+1}, \ldots, f_{j_k} \quad \text{for } k = 1, \ldots, K \tag{2.31}$$

These phrases are used to compute the joint probability for the whole sentence pair:

$$\begin{aligned} Q(f_1^J, e_1^I; s_1^K) = & \prod_{i=1}^{I} [c_1 \cdot p(e_i|e_{i-2}^{i-1})^{\lambda_1}] \\ & \cdot \prod_{k=1}^{K} [c_2 \cdot p(\tilde{f}_k|\tilde{e}_k)^{\lambda_2} \cdot p(\tilde{e}_k|\tilde{f}_k)^{\lambda_3} \\ & \cdot \prod_{j=j_{k-1}+1}^{j_k} p(f_j|\tilde{e}_k)^{\lambda_4} \cdot \prod_{i=i_{k-1}+1}^{i_k} p(e_i|\tilde{f}_k)^{\lambda_5}] \end{aligned} \tag{2.32}$$

With the phrase segmentation s_1^K, the generated phrase lexica $p(\tilde{f}_k|\tilde{e}_k)$, $p(\tilde{e}_k|\tilde{f}_k)$ and the corresponding single word models $p(f_j|\tilde{e}_k)$, $p(e_i|\tilde{f}_k)$ in both directions. This joint probability is used to compute the decision rule which we specified in equation (2.1) in an alternative way:

$$f_1^J \rightarrow (\hat{I}, \hat{e}_1^{\hat{I}}) = \arg\max_{I, e_1^I} \left\{ \max_{K, s_1^K} Q(f_1^J, e_1^I; s_1^K) \right\} \tag{2.33}$$

Log-linear Model Combination

The computation of $Q(f_1^J, e_1^I; s_1^K)$ mentioned above is already a log-linear model combination. A similar method to model the translation probability directly as a log-linear combination is also the usual approach in most state-of-the-art statistical translation systems:

$$p(e_1^I|f_1^J) = \frac{\exp[\sum_m \lambda_m h_m(e_1^I, f_1^J)]}{\sum_{\tilde{e}_1^I} \exp[\sum_m \lambda_m h_m(\tilde{e}_1^I, f_1^J)]} \tag{2.34}$$

with distinct models h_m and corresponding scaling factors λ_m. If we apply *Bayes decision rule* to this log-linear combination, we obtain:

$$f_1^J \rightarrow \hat{e}_1^I(f_1^J) = \arg\max_{e_1^I} \{p(e_1^I|f_1^J)\} = \arg\max_{e_1^I} \{\sum_m \lambda_m h_m(e_1^I, f_1^J)\} \tag{2.35}$$

This gives us a simple way to combine different models aiming for an improved translation quality. Through this, we can easily integrate neural network based language and translation models as well as traditional phrase-based models. The parameters λ_m for all models have to be optimized separately. Usually this is done by optimization methods like *MERT (minimum error rate training)* [Och 03] or *MIRA (margin infused relaxed algorithm)* [Crammer & Singer 03].

Chapter 3

Artificial Neural Networks

The idea of neural networks has been first introduced in 1943 by McCulloch and Pitts [McCulloch & Pitts 43] and had its first renaissance around 1986 [Rumelhart & Hinton$^+$ 86].

Neural networks (NNs) are inspired by the biological structure of the human brain. Simple units which effectively apply a simple, usually non-linear, function σ_j with respect to the *inputs* x_i, a *bias* b_j and the corresponding *weights* w_{ji} are called neurons.

$$z_j = \sum_{i=1}^{I} w_{ji} \cdot x_i + b_j \tag{3.1}$$

$$y_j = \sigma_j(z_j) \tag{3.2}$$

Here σ_j represents an *activation function*, applied to z_j, which is often used to introduce a non-linearity to the computation. Popular choices for activation functions are the *logistic sigmoid*, the *hyperbolic tangent* and the *rectifier* [Nair & Hinton 10] (ReLU: rectified linear unit, NN node with rectifier activation).

$$\sigma_{\text{sigmoid}}(z) = \frac{1}{1 + e^{-z}} \tag{3.3}$$

$$\sigma_{\tanh}(z) = \tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1} \tag{3.4}$$

$$\sigma_{\text{ReLU}}(z) = \max(0, z) \tag{3.5}$$

The choice of the activation function depends on the application and choosing between these activation functions can, in practice, make a difference during optimization [LeCun & Bottou$^+$ 12]. This is necessary, even though the hyperbolic tangent is only a rescaled and shifted sigmoid function:

$$\sigma_{\tanh}(z) = \sigma_{\text{sigmoid}}(2 \cdot z) - 1 \tag{3.6}$$

For training, it is very important that these functions are differentiable, so that the functions composed of them can be optimized by gradient descent (see section 3.4). The first derivatives of σ_{sigmoid} and $\sigma_{\tanh}$ are given by

$$\frac{\partial \sigma_{\text{sigmoid}}(z)}{\partial z} = \sigma_{\text{sigmoid}}(z) \cdot (1 - \sigma_{\text{sigmoid}}(z)) \tag{3.7}$$

$$\frac{\partial \sigma_{\tanh}(z)}{\partial z} = 1 - \tanh(z)^2 \tag{3.8}$$

Since the derivative of σ_{ReLU} would be undefined in $z = 0$, we handle this problem by setting the derivative to

$$\frac{\partial \sigma_{\text{ReLU}}(z)}{\partial z} = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

Neurons are organized in layers with weighted connections between the layers. Such a computation network can be interpreted in various ways. It may be seen as a very simplified simulation of biological brain processes, a way to massively parallelize the computations of a function or one can take the mathematical point of view, as Cybenko did in 1989 [Cybenko 89]. He stated and proved the universal approximation theorem.

Theorem 1 (Universal Approximation Theorem)
Let σ be any continuous, nonconstant, bounded and monotonically-increasing function. Let I_m denote the m-dimensional unit hypercube $[0,1]^m$ or any compact subset of $\mathbb{R}^m$. Then, for $x \in \mathbb{R}^m$, finite sums of the form

$$F(x) = \sum_{j=1}^{N} \gamma_j \sigma(w_j^T x + b_j) \tag{3.10}$$

are dense in $C(I_m)$, the space of continuous functions on I_m.
In other words, given any $f \in C(I_m)$ and $\epsilon > 0$, there is exists an integer N, constants $\gamma_j, b_j \in \mathbb{R}$ and a real vector $w_j \in \mathbb{R}^m$ for $j = 1, \ldots, N$, such that

$$|F(x) - f(x)| < \epsilon \tag{3.11}$$

for all $x \in I_m$, i.e. F is an approximate realization of function f.

If we rewrite formula 3.10, with $N = J$, $m = I$ and $\sigma = \sigma_j \ \forall j = 1, \ldots, J$, we can see that this theorem directly applies to a neural network structure consisting of the components we defined in equation 3.1.

$$F(x) = \sum_{j=1}^{N} \gamma_j \sigma(w_j^T x + b_j) \tag{3.12}$$

$$= \sum_{j=1}^{J} \gamma_j \sigma_j (\sum_{i=1}^{I} w_{ji} x_i + b_j) \tag{3.13}$$

$$= \sum_{j=1}^{J} \gamma_j y_j \tag{3.14}$$

From this, we can conclude that a linear combination of neurons, i.e. an artificial neural network with at least one hidden layer, can approximate any continuous function $f \in C(R^I)$. It is also possible to deduce the same statement for neural networks with multiple outputs.

The universal approximation theorem shows the theoretical effectiveness of artificial neural networks. In practice, this theorem does not help us in finding the parameters we would need to model a given function f. Additionally, the universal approximation theorem gives no statement about whether F would be able to generalize from the approximated function f to different functions $\tilde{f}$. To solve the problem of optimizing parameters w.r.t. a training set, while retaining generalization capability on unseen data, we need efficient methods to find these parameters through training (section 3.4). To make these training methods more efficient and to apply neural networks to special applications, we introduce different structures, such as multilayer perceptrons (section 3.1), convolutional neural networks (section 3.2) and recurrent neural networks (section 3.3).

Remark 1 (Notation)

1. *Time steps and therefore sequence indices are introduced as a parameter $x(t)$ for the corresponding variable x.*
2. *We indicate the affiliation of a variable or parameter to a certain layer l by writing it in the exponent, e.g. $x^{(l)}$ belongs to layer l.*
3. *Parameters w_{ji} correspond to a connection from i to j.*

3.1 Multilayer Perceptron (MLP)

A basic neural network architecture, that is the basis of all following networks, is the *multilayer perceptron (MLP)* [Rumelhart & Hinton$^+$ 86, Werbos 88, Bishop 95]

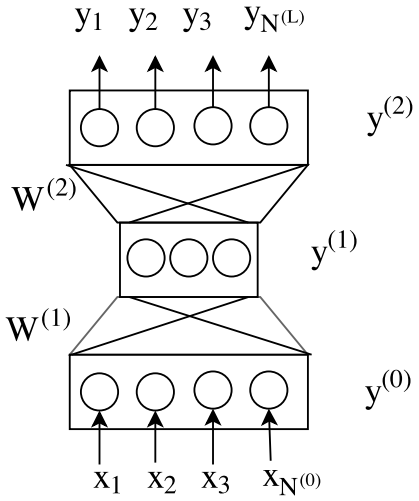

Figure 3.1: Multilayer perceptron with one hidden layer.

which consists of an *input layer*, one or multiple *hidden layers* and an *output layer*, as illustrated in figure 3.1.

An input vector x for such a neural network is presented to the neurons $i = 1, ..., N^{(0)}$ of the input layer 0. The output $y^{(L)}$ would be given by the output of neurons from the output layer after a complete forward pass through all L layers of the neural network. This means that every layer l computes its output from the output $y^{(l-1)}$ of the previous layer $l-1$ by applying the computation specified in formula 3.1 with the parameters of layer l. The same can be achieved by an elementwise application of the activation function $\sigma^{(l)}$ to the product of the weight matrix $W^{(l)} \in \mathbb{R}^{N^{(l)} \times N^{(l-1)}}$ with the input $y^{(l-1)} \in \mathbb{R}^{N^{(l-1)}}$ and an additional bias vector $b^{(l)} \in \mathbb{R}^{N^{(l)}}$. This way we get the following formulas to compute the output of each layer $l = 1, \ldots, L$ if we specify the network input as the output of a virtual layer 0 , i.e. $y^{(0)} := x$.

$$\begin{aligned} g^{(l)}_{\{W^{(l)},b^{(l)}\}} :& \mathbb{R}^{N^{(l-1)}} \to \mathbb{R}^{N^{(l)}}, \\ & y^{(l-1)} \mapsto y^{(l)} = g^{(l)}_{\{W^{(l)},b^{(l)}\}}(y^{(l-1)}) = \sigma^{(l)}(W^{(l)} y^{(l-1)} + b^{(l)}) \end{aligned} \tag{3.15}$$

To simplify this and following formulas, we will be augmenting the input of layer l with a constant to $\overline{y}^{(l)} := [(y^{(l)})^T; 1]^T$ and group all parameters in a single weight matrix $\overline{W}^{(l)} = [W^{(l)}; b^{(l)}]$. Therefore the layer function now looks like this:

$$g^{(l)}_{\overline{W}^{(l)}}(\overline{y}^{(l-1)}) = \sigma^{(l)}(\overline{W}^{(l)}\overline{y}^{(l-1)}) := g^{(l)}_{\{W^{(l)},b^{(l)}\}}(y^{(l-1)}) \tag{3.16}$$

Remark 2 (Notation)
For notational simplicity, the $\overline{\ }$ *will be dropped from here on.*

We can define the whole artificial neural network by concatenating the layer functions, as we defined them above, in the right order.

$$g_\theta : \mathbb{R}^{N^{(0)}} \to \mathbb{R}^{N^{(L)}} : x \mapsto g_\theta(x) = (g^{(L)}_{W^{(L)}} \circ \ldots \circ g^{(1)}_{W^{(1)}})(x) = y^{(L)} \tag{3.17}$$

This new function g_θ is only parameterized on the set of weight matrices $\theta = \{W^{(1)}, \ldots, W^{(L)}\}$. After applying g_θ to the network input x, the final output of the whole network is commonly computed by applying an additional *softmax layer* that is defined to guarantee output normalization:

$$y_c = \frac{e^{y_c^{(L)}}}{\sum_{k=1}^{N^{(L)}} e^{y_k^{(L)}}} \quad \forall\, c \in \{1, \ldots, N^{(L)}\} \tag{3.18}$$

If the network output is normalized like this, it can be interpreted as the class posterior probability if each output unit represents one class c, i.e. the network models

$$p_\theta(c|x) := y_c \tag{3.19}$$

as the probablity distribution defined by the parameter set θ.

3.1.1 Deep Neural Networks (DNNs)

Although one hidden layer would be theoretically sufficient to model most functions, practical experience has shown that neural networks often benefit from multiple hidden layers [LeCun & Bengio$^+$ 15,Mohamed & Dahl$^+$ 12]. Such "deep" structures enable the powerful generalization ability of neural networks and are one of the reasons for the second renaissance of neural networks that we experienced in recent years. To take advantage of these larger structures, it is necessary to find an effective way to train them with acceptable expenses of time and resources. This is a difficult problem, due to the high number of trainable parameters, which leads to the strong tendency of overfitting or the problem that the training gets stuck in poor local optima.

Multiple approaches to solving this problem are categorized as *pre-training* algorithms. There are several supervised pre-training algorithms [Bengio & Lamblin$^+$ 07,Seide & Li$^+$ 11], which start the training by optimizing the network starting with one and iteatively adding more hidden layers during training. There are also some approaches like the *restricted Boltzmann machines (RBM)* [Hinton & Osindero$^+$ 06], which are a form of unsupervised pre-training. For this method, the objective would be to find a representation, which enables an optimal reconstruction of the features in each layer. The hope for these approaches is that the training gets pushed in the direction of a better optimum.

Pre-training might also help to prevent overfitting on the training data and increase the generalization ability, but there are also some methods specifically designed to prevent overfitting. One of the most prominent methods is *dropout* [Hinton & Srivastava$^+$ 12,Srivastava & Hinton$^+$ 14] which artificially reduces the number of parameters during training by randomly ignoring activations of single neurons with probability p.

The greatest development that leads to the recent success of deep neural networks is certainly the increased computation capability. Also, the trend to perform numerical calculations on GPUs resulted in a significant speedup. In fact, numerical libraries [Nickolls & Buck$^+$ 08] which exploit the massive parallelization of GPUs to multi-thread matrix multiplications, can perform computations multiple times faster than a similar implementation on CPU.

Even with all the recent developments that support the training of deep neural networks, it might still be a good idea to use specialized network structures that support parameter sharing up to a certain degree. Examples of such structures that we will discuss in the following are convolutional neural networks (Section 3.2) and recurrent neural network variants (Section 3.3).

3.2 Convolutional Neural Networks (CNNs)

The general concept executed in *convolutional neural networks (CNNs)* was already developed by Kunihiko Fukushima in 1980 under the name ''neocognitron'' [Fukushima 80]. Nine years later, these concepts were successfully applied to handwritten character recognition by Le Cun [LeCun & Boser$^+$ 89]. Today, the architectures are basically the same as 27 years ago, with the important difference that the networks have become much larger with over 30 layers [Szegedy & Liu$^+$ 15, Krizhevsky & Sutskever$^+$ 12], which enables these networks to outperform traditional methods in image recognition and other tasks.

The basic idea is inspired by the visual cortex of animals [Hubel & Wiesel 68]. In this biological background, there are specific cells that act as local filters over the input space. In this fashion, CNNs apply relatively small weight matrices

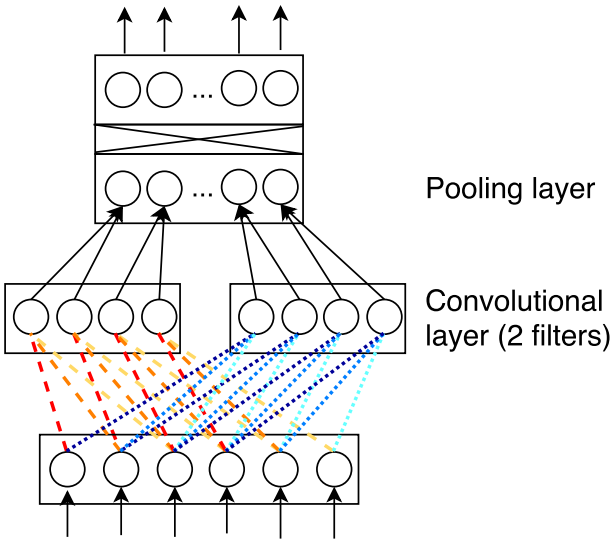

Figure 3.2: Convolutional neural network with $k = 1$ and 2 filters.

$W^{(l)} \in \mathbb{R}^{1\times(2\cdot k^{(l)}+1)}$ on parts of the input and shift these over the whole input to compute the layer output, by calculating

$$y_j^{(l)} = \sigma^{(l)}\Big(\sum_{i=j-k^{(l)}}^{j+k^{(l)}} w_{j-i}^{(l)} \cdot y_i^{(l-1)}\Big) \tag{3.20}$$

for all $j \in 1, \ldots, N^{(l)}$. Corresponding to the biological inspiration, the shifted matrices are also called *filter* or *kernel*. The size of the receptive fields that are covered by these filters is defined by $k^{(l)}$ and is one important aspect that needs to be empirically optimized to find the best possible network architecture. It is also possible to use multiple filters in parallel, due to the relatively low dimensionality of the filters and the high amount of parameter sharing.

Defined as above, the operation is very similar to the convolution operation

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{3.21}$$

which is defined in signal processing for two continuous functions f and g.

Using this definition, we can rewrite equation (3.20) as the convolution of $w^{(l)}$ and $y^{(l-1)}$:

$$y_j^{(l)} = \sigma^{(l)}((w^{(l)} * y^{(l-1)})(j)) \tag{3.22}$$

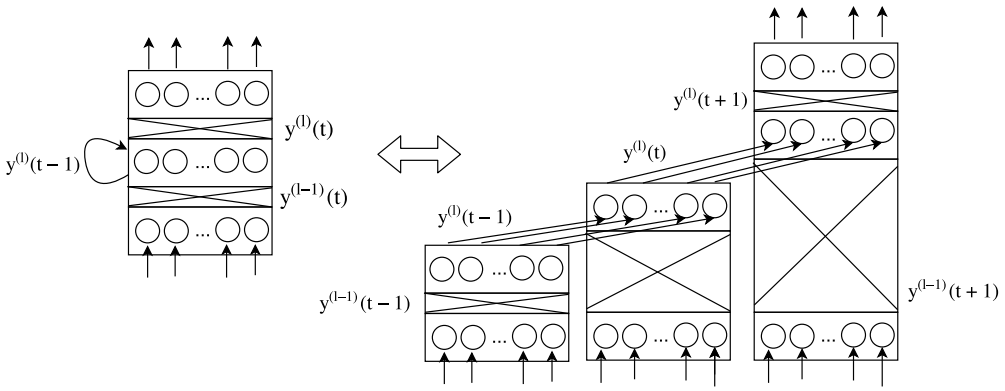

Figure 3.3: RNN with its equivalent unfolded in time for three time steps.

To further reduce the dimensionality for layers deeper in the network, it is possible to introduce a pooling layer which takes the maximum (*max-pooling*)

$$y_j^{(l')} = \max_{i=j-k^{(l')},\ldots,j,\ldots,j+k^{(l')}} \{y_i^{(l)}\} \tag{3.23}$$

or the mean (*mean-pooling*)

$$y_j^{(l')} = \frac{1}{2 \cdot k^{(l')} + 1} \sum_{i=j-k^{(l')}}^{j+k^{(l')}} y_i^{(l)} \tag{3.24}$$

from a window of size $2 \cdot k^{(l')} + 1$. In the traditional applications of such convolutional architectures, the formulas were derived for two dimensional data, as they would be needed for image recognition [Szegedy & Liu$^+$ 15, Krizhevsky & Sutskever$^+$ 12] or related tasks [Xu & Ba$^+$ 15, Ren & Kiros$^+$ 15]. But the formulas can be adapted to input data of different dimensionality, as shown above for one-dimensional input data. The application of such CNNs for one-dimensional data, like text, was demonstrated successfully in several examples [dos Santos & Zadrozny 14, Hu & Lu$^+$ 14] and is executed for our use-case in chapter 5.

3.3 Recurrent Neural Networks (RNNs)

Feed forward neural networks are very popular and can be extremely powerful [LeCun & Bengio$^+$ 15], but they lack certain properties which are necessary for

various tasks. Feed forward neural networks depend on a fixed input size, which results from the fixed dimensionality of their input layer. They also are not aware of the context of their input, i.e. the previous or following input. This is all due to the fact that feed forward neural networks are not capable of memorizing any information, such as the previous input.

This problem is solved by introducing cyclic connections for the hidden layers of a neural network [Elman 90, Jordan 86, Lang & Waibel$^+$ 90]. Such a network is called a *recurrent neural network (RNN)*. At every time step t, the hidden layers of an RNN get, additionally to the input from the previous layer $y^{(l-1)}(t)$, their own output $y^{(l)}(t-1)$ from the previous time step as a second input, weighted by a separate weight matrix $U^{(l)} \in \mathbb{R}^{N^{(l)} \times N^{(l)}}$.

$$y^{(l)}(t) = \sigma^{(l)}(W^{(l)} y^{(l-1)}(t) + U^{(l)} y^{(l)}(t-1)) \tag{3.25}$$

RNNs can be seen as a arbitrary deep MLP, as it is possible to unfold the RNN over the time axis, which is illustrated in figure 3.3. This means that every time step could also be viewed as an additional layer in the whole network with its own input and connections to the layer of the previous time step, but with shared weight parameters on these links. Using this perspective, it becomes evident that RNNs are another example of a deep network structure since the network can theoretically be arbitrary deep in time.

3.3.1 Long Short-Term Memory (LSTM)

RNNs with a naive architecture as described before, have proven not to be very successful in storing information of previous input over long time distances. The reason for this is a problem called *vanishing gradient* [Hochreiter 91, Hochreiter & Bengio$^+$ 01, Bengio & Simard$^+$ 94]. This is caused by applying the chain rule multiple times and thereby multiplying gradients smaller than 1. Therefore, the gradient of time steps lying further in the past approaches a value of zero with exponential speed. Another possibility would be the *exploding gradient problem*, i.e. it is possible that the gradient blows up exponentially if the gradients that get multiplied are greater than 1. One simple solution to the latter problem would be to set an upper bound for the gradient. This technique is called gradient clipping. The downside to this method would be a loss of possibly important information that was contained in the clipped gradient, which does not seem to have a great effect in practice. Another disadvantage of the standard RNN architecture can be the fact that the memory, i.e. the hidden state which is handed over through time, is not protected and gets manipulated in every time step.

A solution for that solves the vanishing gradient problem and protects the memory at the same time was proposed by Hochreiter and Schmidhuber in 1997 [Hochreiter

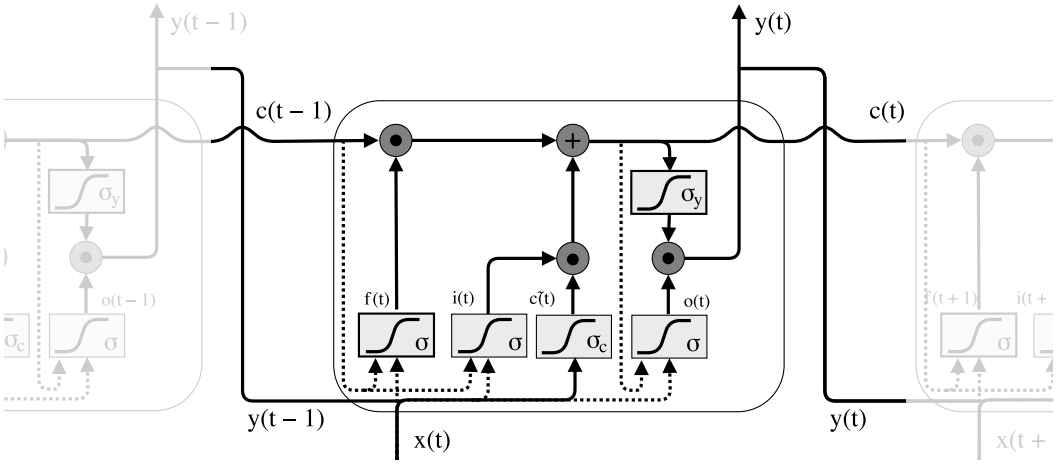

Figure 3.4: Architecture of an LSTM unfolded over time.[1]

& Schmidhuber 97a], who developed the *long short-term memory (LSTM)* architecture. An LSTM block describes a unit that can be used instead of a standard neuron. It contains the recurrent information in a separate, protected, state which can be manipulated explicitly. Therefore an LSTM block gets, additionally to the output from the previous layer $x(t)$ and the output of the previous time step $y(t-1)$, the state of the cell $c(t-1)$ as input, which was also calculated in the previous time step. The architecture of an LSTM unit is illustrated in figure 3.4, where it is drawn with its connections unfolded over time.

To enable the manipulation of the cell state and the output to the next layer, while protecting the cell state at the same time, Hochreiter introduced three gating components, namely the *forget* (f), *input* (i) and *output gate* (o), which are computed from all three inputs in the following way:

$$f(t) = \sigma(W_{xf}x(t) + W_{yf}y(t-1) + W_{cf}c(t-1) + b_f) \tag{3.26}$$
$$i(t) = \sigma(W_{xi}x(t) + W_{yi}y(t-1) + W_{ci}c(t-1) + b_i) \tag{3.27}$$
$$o(t) = \sigma(W_{xo}x(t) + W_{yo}y(t-1) + W_{co}c(t) + b_o) \tag{3.28}$$

The only differences between these gates are the parameters designated to each gate and the fact that the output gate $o(t)$ depends on the current cell state $c(t)$ instead of the one from the previous timestep. This is due to the fact that the output gate is computed after the cell was already updated. A logistic sigmoid function is a

[1] Graphic inspired by `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

common choice for the activation function σ so that the gate activations are in a range from 0 to 1 to act as a closed or opened gate.

Utilizing these gating components, the new cell state, and unit output are computed. For this, we first compute an update candidate, by applying a non-linear activation function and weight matrices to the output from the previous timestep and the previous layer:

$$\tilde{c}(t) = \sigma_c(W_{xc}x(t) + W_{yc}y(t-1) + b_c) \tag{3.29}$$

The output $f(t)$ of the *forget gate* is multiplied componentwise with the old state $c(t-1)$ of the cell. This implies that a value of 0 in $f(t)$ resets the state of the cell, i.e. previous information is deleted. The *input gate* output $i(t)$ decides if the *update candidate* $\tilde{c}(t)$, which is calculated by function σ_c, is updated or not. Therefore, the activation of the input gate $i(t)$ is multiplied componentwise with $\tilde{c}(t)$ and afterwards added to the old state that was filtered through the forget gate.

$$c(t) = f(t) \odot c(t-1) + i(t) \odot \tilde{c}(t) \tag{3.30}$$

Finally the *output gate* decides which parts of the output, calculated from the new state $c(t)$ by an activation function σ_y are propagated as the new output.

$$y(t) = o(t) \odot \sigma_y(c(t)) \tag{3.31}$$

The activation functions σ_c, σ_y for the cell input and output are usually *tanh* or logistic sigmoid, but σ_y can also be the identity in some cases.

The important attribute of LSTMs, which enables them to preserve information over long time distances, is the cell state which is updated without applying an activation directly. Therefore we can compute the gradient as the derivative of the identity function, which is 1 and leads to a stable gradient flow that will not vanish over time.

Whether an LSTM or any of its variants is used, the significant advantage over standard RNNs is its ability to remember information over a longer period of time. This longer memory can be advantageous in many cases. Obviously, long range and disconnected patterns like quotes or opening and closing brackets benefit from a longer memory capability. In general, it has been shown that longer memory improves a system's prediction capability [**?**], which will be useful for the applications in the following chapters.

3.3.2 Gated Recurrent Unit (GRU)

In the last years, there have been many publications that introduced numerous variants of LSTMs [Yao & Cohn$^+$ 15, **?**]. One variant is the Gated Recurrent Unit (GRU) which was introduced by Cho et al. in 2014 [Cho & van Merrienboer$^+$

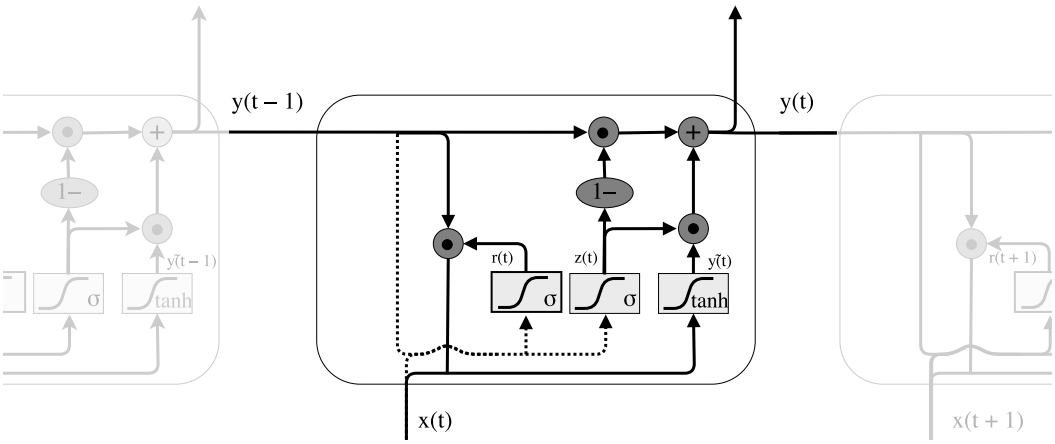

Figure 3.5: Architecture of a GRU unfolded over time.[2]

14, Chung & Gulcehre$^+$ 14]. The greatest change from the LSTM to the GRU architecture is the fact that the cell and output state are replaced by a single activation state $y(t)$ for the GRU architecture. Also, the input and forget gates are combined to a single update gate $z(t)$ which is used to compute the new activation as a linear interpolation between the previous activation $y(t-1)$ and the candidate activation $\tilde{y}(t)$

$$y(t) = [1 - z(t)] \odot y(t-1) + z(t) \odot \tilde{y}(t) \tag{3.32}$$

The candidate activation depends on the current input $x(t)$ and the output from the previous timestep $y(t-1)$, filtered by the reset gate $r(t)$.

$$\tilde{y}(t) = \sigma_{\text{tanh}}(W_{xy}x(t) + W_{yy}[r(t) \odot y(t-1)] + b_y) \tag{3.33}$$

The update and reset gates are computed similarly to the gate components of an LSTM unit.

$$r(t) = \sigma_{\text{sigmoid}}(W_{xr}x(t) + W_{yr}y(t-1) + b_r) \tag{3.34}$$
$$z(t) = \sigma_{\text{sigmoid}}(W_{xz}x(t) + W_{yz}y(t-1) + b_z) \tag{3.35}$$

All these changes compared to the LSTM unit result in a reduction in the number of parameters, to two thirds of the parameters an LSTM would have. This makes GRUs easier to compute while at the same time being equally powerful as LSTMs on sequence modeling tasks [Chung & Gulcehre$^+$ 14].

[2]Graphic inspired by `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

3.4 Training Neural Networks

Neural networks can be very powerful models with millions of parameters, which need to be optimized. For this reason, neural network training is a hard problem, which slowed down the progress with neural networks for many years.

The advantage of neural networks is that all parameters in the network can be trained jointly, with respect to a loss function that needs to be chosen depending on the task at hand. Since nowadays almost all neural networks are trained by some gradient descent algorithm, it is also necessary to compute the respective gradient in an efficient way, to be able to update the weights w.r.t. the gradient.

3.4.1 Loss Functions

There are different ways to handle neural network output. In the following, we will focus on interpreting the neural network as a classifier that tries to predict the true output class for a given input [Ney 16].

Assuming we have a set of annotated training data in way that for each input vector x_n we know the correct output class c_n, i.e.

$$\mathcal{X}_{\text{train}} = \{(x_n, c_n)|\ n = 1, \ldots, N\} \tag{3.36}$$

Let us further assume we have a neural network model $p_\theta(c, x)$, with free parameters θ, that generates an output for every c in the set of possible classes $\mathcal{C}$.

Squared Error Criterion

Our goal is to achieve an ouput of 1 for the correct class, i.e. $p_\theta(c = c_n, x_n) = 1$, and 0 for all other classes, i.e. $p_\theta(c \neq c_n, x_n) = 0$. One possibility to train such a behavior without enforcing any constraints on the neural network output would be the *mean squared error (MSE)* criterion:

$$\mathcal{L}_{\text{MSE}} := \frac{1}{N} \sum_n \sum_{c \in \mathcal{C}} [p_\theta(x_n, c) - \delta(c, c_n)]^2 \tag{3.37}$$

where $\delta(c, c_n) = 1$ if and only if $c_n = c$, else 0.

Cross Entropy Criterion

For the case that we want to model a probability distribution with the network output, i.e. the neural network model is now $p_\theta(c|x)$, and we have the constraints

$$p_\theta(c|x) \geq 0 \quad \text{and} \quad \sum_{c \in \mathcal{C}} p_\theta(c|x) = 1.0 \tag{3.38}$$

we can use the cross entropy criterion as our loss function. For obvious reasons, this is also called negative log probability:

$$\mathcal{L}_{\text{CE}} := -\frac{1}{N}\sum_{n} \log p_\theta(c_n|x_n) \tag{3.39}$$

By using the empirical distribution $pr(c,x) = \frac{1}{N}\sum_{n=1}^{N}\delta(c,c_n)\delta(x,x_n)$ of the training data $\mathcal{X}_{\text{train}}$, where δ is the Kronecker delta, we can rewrite it as following:

$$\mathcal{L}_{\text{CE}} = -\sum_{x\in\mathcal{X},c\in\mathcal{C}} pr(c,x)\log p_\theta(c|x) \tag{3.40}$$

$$= -\sum_{x\in\mathcal{X}} pr(x)\sum_{c\in\mathcal{C}} pr(c|x)\log p_\theta(c|x) \tag{3.41}$$

Since we know that the empirical distribution of x is represented by $pr(x) = \frac{1}{N}\sum_{n=1}^{N}\delta(x,x_n)$, it becomes obvious that the defined loss function is really the cross-entropy between the empirical distribution $pr(c|x)$ and our model distribution $p_\theta(c|x)$.

$$\mathcal{L}_{\text{CE}} = -\sum_{x\in\mathcal{X}}\frac{1}{N}\sum_{n=1}^{N}\delta(x,x_n)\sum_{c\in\mathcal{C}} pr(c|x)\log p_\theta(c|x) \tag{3.42}$$

$$= -\frac{1}{N}\sum_{n=1}^{N}\sum_{c\in\mathcal{C}} pr(c|x_n)\log p_\theta(c|x_n) \tag{3.43}$$

3.4.2 Weight Update

In training, we want to minimize the loss function we chose for the network. To achieve this, it is necessary to adjust every single weight parameter W by the right amount. Since a neural network is, by definition, a composition of differentiable functions, it is possible to compute the gradient with respect to a certain parameter. The gradient intuitively points in the direction of the steepest ascent, so following the negative error gradient will automatically lead to a minimum, which will not necessarily be a global minimum. This is why the naive way of training simply takes a small, fixed-size step in the direction of the negative error gradient of the loss function:

$$w_{ji}^{(l),n+1} = w_{ji}^{(l),n} + \Delta w_{ji}^{(l),n} \tag{3.44}$$

$$\Delta w_{ji}^{(l),n} = -\eta\frac{\partial\mathcal{L}}{\partial w_{ji}^{(l),n}} \tag{3.45}$$

where $\Delta w_{ji}^{(l),n}$ is the n^{th} weight update applied to the weight parameter $w_{ji}^{(l)}$ and $\eta \in [0, 1]$ is the *learning rate* that controls the "speed" of the descent. The biggest problem with this method is that it easily gets stuck in local minima. It is common to add a momentum term to prevent this:

$$\Delta w_{ji}^{(l),n} = \mu \Delta w_{ji}^{(l),n-1} - \eta \frac{\partial \mathcal{L}}{\partial w_{ji}^{(l),n}} \tag{3.46}$$

where $\mu \in [0, 1]$ is the momentum parameter that is multiplied with the update from the previous update step $\Delta w_{ji}^{(l),n-1}$. This additional term might speed up convergence and help to escape local minima. This method of updating weights by taking steps in the direction of the negative gradient is called *gradient descent*.

Applying gradient descent iteratively for every example in the training set is known as *online learning* or *stochastic gradient descent (SGD)*. In contrast to that stands *batch learning*, where the gradient is computed, and the weight updates are applied w.r.t. the whole training set. A compromise between those two extremes would be to compute the gradient for mini-batches, representing a fraction of the training data, which is commonly used in practice.

Adaptive Learning Rate Control: Adadelta

Even with momentum parameter and mini-batch updates, it often happens, especially with sparse data, that convergence is very slow, or the algorithm gets stuck in local minima. This can often be prohibited by adapting the learning rate during training. However, doing so by hand would not be feasible in most cases. A solution to this problem is offered by algorithms like *adagrad* [Duchi & Hazan$^+$ 11], which adapts the learning rate to the parameters. Therefore, it performs smaller updates on frequently changed parameters and larger updates on infrequently changed parameters. This way, the multiple occurrences of a certain value in the training data result in much smaller updates than for values that only occur once. That makes the algorithm a good fit for sparse training data.

Adadelta [Zeiler 12] is an extension of adagrad, that tries to prevent the monotonic shrinking of the learning rate during training, which is the main problem for adagrad. For this, we replace the fixed learning rate η in formula (3.45) by the

root mean squared error (RMS) of parameter updates divided by the RMS of the gradient:[3]

$$\Delta\theta_n = -\frac{\sqrt{E[\Delta\theta^2]_n + \epsilon}}{\sqrt{E[g^2]_n + \epsilon}} \underbrace{\frac{\partial \mathcal{L}}{\partial \theta_n}}_{=:g_n} \tag{3.47}$$

$$= -\frac{\mathrm{RMS}[\Delta\theta]_n}{\mathrm{RMS}[g^2]_n} g_n \tag{3.48}$$

Here ϵ is a smoothing term in the order of 10^{-8} and $E[g^2]_n$ and $E[\Delta\theta^2]_n$ are running averages over all past squared gradients respectively all past squared parameter updates:

$$E[g^2]_n = \gamma E[g^2]_{n-1} + (1-\gamma) g_n^2 \tag{3.49}$$

$$E[\Delta\theta^2]_n = \gamma E[\Delta\theta^2]_{n-1} + (1-\gamma) \Delta\theta_n^2 \tag{3.50}$$

where γ is a value similar to the momentum term, that should be set to a value around 0.9.

3.4.3 Backpropagation

For gradient descent to work, it is important to be able to compute the gradient in an efficient way. To enable this for neural networks with one or multiple hidden layers, the standard technique used in most cases is *backpropagation* [Rumelhart & Hinton$^+$ 86, Werbos 88, Williams & Zipser 95]. This method is basically a repeated application of the chain rule for partial derivatives.

To compute the derivative with respect to a single weight matrix $W^{(l)}$, we can just compute the derivative w.r.t. every single matrix component separately.

$$\frac{\partial \mathcal{L}}{\partial W^{(l+1)}} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial w_{11}^{(l+1)}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{1N^{(l)}}^{(l+1)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial w_{N^{(l+1)}1}^{(l+1)}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{N^{(l+1)}N^{(l)}}^{(l+1)}} \end{bmatrix} \tag{3.51}$$

And since we know the chain of dependencies in our network looks like this

$$\mathcal{L} \to y^{(L)} \to \ldots \to y^{(l+2)} \to y^{(l+1)} \begin{array}{l} \nearrow y^{(l)} \to \ldots \to y^{(1)} \to x \\ \searrow W^{(l+1)} \end{array} \tag{3.52}$$

[3] For notational simplicity, we replace the explicit weight component $w_{ji,n}^{(l)}$ by the general formulation θ_n.

we can simply apply the chain rule to first compute the derivative of $\mathcal{L}$ with respect to $y_i^{(l)}$

$$\Delta_i^{(l)} := \frac{\partial \mathcal{L}}{\partial y_i^{(l)}} = \sum_{j=1}^{N^{(l+1)}} \frac{\partial \mathcal{L}}{\partial y_j^{(l+1)}} \frac{\partial y_j^{(l+1)}}{\partial y_i^{(l)}} \tag{3.53}$$

$$= \sum_{j=1}^{N^{(l+1)}} \Delta_j^{(l+1)} \frac{\partial y_j^{(l+1)}}{\partial y_i^{(l)}} \tag{3.54}$$

$$= \sum_{j=1}^{N^{(l+1)}} \Delta_j^{(l+1)} \sigma^{(l+1)\prime}(z_j^{(l+1)}) w_{ji}^{(l+1)} \tag{3.55}$$

where $y_i^{(l)}$ is the i^{th} output of layer l. Executing this recursive equation for all $l = L, \ldots, 0$ and $i = 1, \ldots, N^{(l)}$ in a backward pass until the we reach

$$\Delta_i^{(0)} = \frac{\partial \mathcal{L}}{\partial y_i^{(0)}} = \sum_{j=1}^{N^{(1)}} \Delta_j^{(1)} \frac{\partial y_j^{(1)}}{\partial x_i} \tag{3.56}$$

gives us all the components we need to compute the derivative w.r.t. weight component $w_{ji}^{(l)}$ by once more applying the chain rule:

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}} = \frac{\partial \mathcal{L}}{\partial y_j^{(l)}} \frac{\partial y_j^{(l)}}{\partial w_{ji}^{(l)}} = \Delta_j^{(l)} \frac{\partial y_j^{(l)}}{\partial w_{ji}^{(l)}} \tag{3.57}$$

What is left to compute here is

$$\frac{\partial y_j^{(l)}}{\partial w_{ji}^{(l)}} = \frac{\partial \sigma^{(l)}(z_j^{(l)})}{\partial z_j^{(l)}} \frac{\partial \sum_{h=1}^{N^{(l-1)}} w_{jh}^{(l)} y_h^{(l-1)}}{\partial w_{ji}^{(l)}} \tag{3.58}$$

$$= \sigma^{(l)\prime}(z_j^{(l)}) \frac{\partial w_{ji}^{(l)} y_i^{(l-1)}}{\partial w_{ji}^{(l)}} \tag{3.59}$$

$$= \sigma^{(l)\prime}(z_j^{(l)}) \cdot y_i^{(l-1)} \tag{3.60}$$

for all $j = 1, \ldots, N^{(l)}$. Plugging this into equation (3.57), we finally have

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}} = \Delta_j^{(l)} \cdot \sigma^{(l)\prime}(z_j^{(l)}) \cdot y_i^{(l-1)} \tag{3.61}$$

where $\sigma^{(l)\prime}$ is the derivative of the activation in layer l. The activation function is typically one of the three we mentioned at the beginning of this chapter and therefore its derivative is specified by equations (3.7) to (3.9).

Also unspecified in the derivation up to this point is the starting point of the backward-pass, $\Delta_i^{(L)}$, which is dependent on the choice of loss function and output layer.

We will now derive $\Delta_i^{(L)}$ for the cross-entropy loss function which is the most common loss function for the training of networks with a softmax output and will be used throughout this thesis. The derivation for different loss functions works analogously. A softmax layer is commonly used as the output layer L to match the normalization constraints and take advantage of a discriminative approach at the same time. This way we first need to plug equation (3.18) into equation (3.39) to get the explicit loss of our network:

$$\mathcal{L} = -\frac{1}{N}\sum_n \log p_\theta(c_n|x_n) \tag{3.62}$$

$$= -\frac{1}{N^{(L)}}\sum_{n=1}^{N^{(L)}} \log\left[\frac{e^{y_{c_n}^{(L)}}}{\sum_{k=1}^{N^{(L)}} e^{y_k^{(L)}}}\right] \tag{3.63}$$

$$= -\frac{1}{N^{(L)}}\sum_{n=1}^{N^{(L)}} \left[y_{c_n}^{(L)} - \log(\sum_{k=1}^{N^{(L)}} e^{y_k^{(L)}})\right] \tag{3.64}$$

If we further take the derivative w.r.t. $y_i^{(L)}$, we get $\Delta_i^{(L)}$:

$$\Delta_i^{(L)} = \frac{\partial \mathcal{L}}{\partial y_i^{(L)}} \tag{3.65}$$

$$= -\frac{1}{N^{(L)}}\left(\left[\frac{1}{\partial y_i^{(L)}}\sum_{n=1}^{N^{(L)}} y_{c_n}^{(L)}\right] - \left[\frac{1}{\partial y_i^{(L)}}\sum_{n=1}^{N^{(L)}} \log(\sum_{k=1}^{N^{(L)}} e^{y_k^{(L)}})\right]\right) \tag{3.66}$$

$$= -\frac{1}{N^{(L)}} + \frac{1}{N^{(L)}}\left[\frac{N^{(L)}}{\sum_{k=1}^{N^{(L)}} e^{y_k^{(L)}}}\frac{\sum_{k=1}^{N^{(L)}} e^{y_k^{(L)}}}{\partial y_i^{(L)}}\right] \tag{3.67}$$

$$= -\frac{1}{N^{(L)}} + \frac{e^{y_i^{(L)}}}{\sum_{k=1}^{N^{(L)}} e^{y_k^{(L)}}} \tag{3.68}$$

Therefore all necessary components are defined to perform the recursive computation of the weight derivatives in the following way:

1. Compute $y_i^{(l)}$ for all $l \in 1, \ldots, L$ and $i \in 1, \ldots, N^{(l)}$ in a forward pass through the network.

2. Use these values to compute the starting point for the recursion $\Delta_i^{(L)}$ with the derived equation (3.68).

3. Given $\Delta_i^{(L)}$, recursively compute $\Delta_i^{(l)}$ for all $l = L-1, \ldots, 0$ and $i = 1, \ldots, N^{(l)}$ by following equation (3.53) and $\Delta_i^{(0)}$ defined by equation (3.56).
4. Finally, compute all $\frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}}$ in a second backward-pass, by applying equation (3.57).

This way we compute all components of $\frac{\partial \mathcal{L}}{\partial W^{(l)}}$ for all $l = 1, \ldots, L$ and are able to perform the weight update for the whole network.

Backpropagation through Time

To use gradient descent on RNNs, we have to consider some particular challenges. Therefore *real time recurrent learning* [Robinson & Fallside 87] and *backpropagation through time* (BPTT) [Williams & Zipser 95, Werbos 90] have been developed to efficiently calculate the derivative with respect to the weights for RNNs. BPTT is more efficient than real time recurrent learning and works in principle very similar to standard backpropagation. Where BPTT differs from the standard algorithm is that the loss function depends not only on the activation of the hidden layer at time step t, but also on the activation of the previous time steps. Therefore also the recursive computation needs to be time dependent.

$$\Delta_i^{(l)}(t) := \frac{\partial \mathcal{L}}{\partial y_i^{(l)}(t)} \tag{3.69}$$

$$= \sum_{j=1}^{N^{(l+1)}} \Delta_j^{(l+1)}(t) \frac{\partial y_j^{(l+1)}(t)}{\partial y_i^{(l)}(t)} + \sum_{k=1}^{N^{(l)}} \Delta_k^{(l)}(t+1) \frac{\partial y_k^{(l)}(t+1)}{\partial y_i^{(l)}(t)} \tag{3.70}$$

The derivatives for the complete sequence can therefore be calculated by a recursive computation of $\Delta_i^{(l)}(t)$ not only for all $l = L, \ldots, 0$ and $i = 1, \ldots, N^{(l)}$ but also for all $t = T, \ldots, 1$ starting at T and going backwards in time from there. Given the fact that the weight parameters are the same for each time step, it is easy to see that the derivative with respect to the weight can be obtained by summation over all time steps:

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}} = \sum_{t=1}^{T} \Delta_j^{(l)}(t) \frac{\partial y_j^{(l)}(t)}{\partial w_{ji}^{(l)}} \tag{3.71}$$

To train the parameters of a network that is composed of LSTM cells, it is very convenient that the computation to calculate the output is again just a composition of functions. Therefore, it is possible to apply the chain rule and use backpropagation through time and stochastic gradient descent directly without

any modification. The corresponding equations can be found in the original paper [Hochreiter & Schmidhuber 97a] or in the dissertation of Alex Graves [Graves 12], where they are derived in a similar style to the equations listed above. In the original LSTM approach [Hochreiter & Schmidhuber 97a] the backpropagation through time algorithm was even truncated after one time step, although this is not always necessary [Graves & Schmidhuber 05].

Backpropagation for Convolution

If we want to perform backpropagation on CNNs, we can take advantage of the fact that a convolutional layer is only one function that is part of a chain of many concatenated functions that form the neural network. Therefore all we need to do if layer l is a convolutional layer, is to compute $\Delta_i^{(l-1)}$ for $i = 1, \dots, N^{(l-1)}$ and $\frac{\partial \mathcal{L}}{\partial w_c^{(l)}}$ for $c = -k^{(l)}, \dots, k^{(l)}$. To derive such a $\Delta_i^{(l-1)}$ as specified in (3.53), for the special case of a convolutional layer as we defined it in (3.22) we have the following derivation:

$$\Delta_i^{(l-1)} = \frac{\partial L}{\partial y_i^{(l-1)}} = \sum_{j=1}^{N^{(l)}} \Delta_j^{(l)} \frac{\partial y_j^{(l)}}{\partial y_i^{(l-1)}} \tag{3.72}$$

$$= \sum_{j=1}^{N^{(l)}} \Delta_j^{(l)} \sigma'(z_j^{(l)}) \frac{\partial z_j^{(l)}}{\partial y_i^{(l-1)}} \tag{3.73}$$

$$= \sum_{j=1}^{N^{(l)}} \Delta_j^{(l)} \sigma'(z_j^{(l)}) \frac{\partial \sum_{m=j-k^{(l)}}^{j+k^{(l)}} w_{j-m}^{(l)} \cdot y_m^{(l-1)}}{\partial y_i^{(l-1)}} \tag{3.74}$$

$$= \sum_{j=1}^{N^{(l)}} \Delta_j^{(l)} \sigma'(z_j^{(l)}) w_{j-i}^{(l)} \tag{3.75}$$

Similarly, the derivative with respect to a single weight parameter is derived by plugging definition (3.22) into definition (3.57):

$$\frac{\partial \mathcal{L}}{\partial w_c^{(l)}} = \sum_{j=1}^{N^{(l)}} \frac{\partial \mathcal{L}}{\partial y_j^{(l)}} \frac{\partial y_j^{(l)}}{\partial w_c^{(l)}} = \sum_{j=1}^{N^{(l)}} \Delta_j^{(l)} \frac{\partial y_j^{(l)}}{\partial w_c^{(l)}} \tag{3.76}$$

$$= \sum_{j=1}^{N^{(l)}} \Delta_j^{(l)} \sigma'(z_j^{(l)}) \frac{\partial \sum_{m=j-k^{(l)}}^{j+k^{(l)}} w_{j-m}^{(l)} \cdot y_m^{(l-1)}}{\partial w_c^{(l)}} \tag{3.77}$$

$$= \sum_{j=1}^{N^{(l)}} \Delta_j^{(l)} \sigma'(z_j^{(l)}) y_{j-c}^{(l-1)} \tag{3.78}$$

Chapter 4

Neural Machine Translation

The task of generating sequences is useful in many practical applications like recognition or synthesis for speech and handwriting and also in machine translation. To solve this problem, it is possible to use a specific neural network architecture, which can be trained end-to-end by backpropagation. The application of such neural networks to the machine translation problem defines the field of *neural machine translation*.

To apply these general end-to-end models to MT, the network needs to handle words and special *tokens*, like a *sentence-end-token* that indicates the end of a sentence, as input and output. Text data as a discrete input or output data can be represented using *"one hot" encoding*, i.e. V text classes result in a vector with V entries where for class v only entry v is set to one while the others are filled with zeros. The text classes are usually all possible words in the vocabulary $\mathcal{V} = \{\omega_1, \ldots, \omega_V\}$ with V entries or, if modeling happens on the character level, all possible characters. The necessary multinomial distribution of V classes that we need to represent the whole vocabulary in the output layer can be obtained by using a softmax function for the activation of the output layer.

If the modeling happens on the word level, it is also possible to use vectors generated from word embedding [Bengio & Ducharme$^+$ 03] as input or output of the network. This means using a parameterized function to map words to high-dimensional vectors, where semantically similar vectors are mapped to similar vectors. This parameterized function is usually integrated into the network as an input layer with V input units which is connected to the next layer through a linear mapping on a much smaller dimension. The parameters of this linear mapping will be learned by backpropagation as part of the neural network training.

4.1 Encoder-Decoder Approach

The first sequence-to-sequence models applied to neural machine translation that showed competitive results to phrase-based machine translation systems were proposed in 2014 by Sutskever et al. [Sutskever & Vinyals$^+$ 14] and Cho et al. [Cho & van Merrienboer$^+$ 14]. Both models are not limited to neural machine translation,

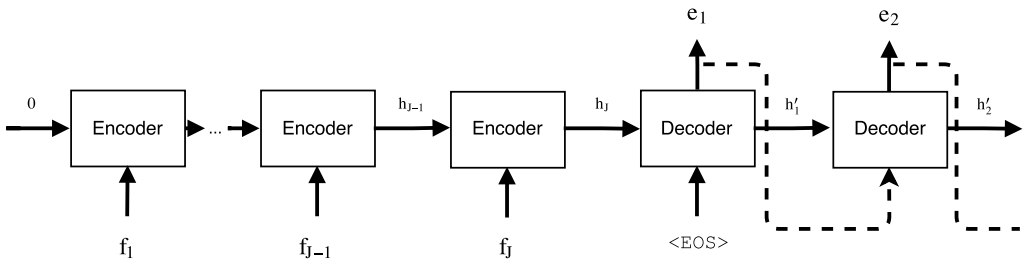

Figure 4.1: The RNN-Encoder and Decoder Model [Sutskever & Vinyals$^+$ 14]

because of their very general architecture that can be applied to various sequence-to-sequence generation tasks without any greater changes. Since both models are quite similar in their structure and resulted in a similar translation performance, we will focus on the approach by Sutskever et al. [Sutskever & Vinyals$^+$ 14] in the following.

The architecture consists of two separate RNNs. The first RNN, so called *encoder*, takes the given source sentence $f_1^J = f_1, \ldots, f_J$ one word at a time and encodes it to a fixed size vector, the *context* or *thought vector* c, that is automatically generated as the hidden state of the RNN. When the source sentence is terminated by the special `<EOS>` token, the thought vector is handed over to the second RNN, i.e. the hidden state of this *decoder* RNN is set to the last hidden state of the encoder RNN. Afterwards the decoder RNN starts generating the output sentence $e_1^I = e_1, \ldots, e_I$.

$$\Pr(e_1, \ldots, e_I | f_1, \ldots, f_J) = \prod_{i=1}^{I} \mathrm{p}(e_i | c, e_1, \ldots, e_{i-1}) \tag{4.1}$$

The corresponding loss function is given by the cross-entropy of a correct sentence e_1^I given its source sentence f_1^J.

$$\mathcal{L}(e_1^I, f_1^J) = -\log \Pr(e_1^I | f_1^J) = -\sum_{i=1}^{I} \log \mathrm{p}(e_i | c, e_1^{i-1}) \tag{4.2}$$

Since, for reasons explained in section 3.3.1, this simple approach is hard to train with standard RNNs, Sutskever et al. [Sutskever & Vinyals$^+$ 14] used LSTMs instead. Moreover, they separated the encoder and decoder into two LSTMs which do not share any parameters but will be trained jointly. In their paper, they used this LSTM architecture with four layers, word vectors as input and a softmax function

over all words in the vocabulary to predict the output word. This way, the model generates a probability distribution for each decoder timestep. To find the most likely translation for a given input sentence, Sutskever et al. [Sutskever & Vinyals$^+$ 14] suggest a simple left-to-right beam search decoding. That means that for each decoding step, we extend a set of B partial hypotheses with every possible word in the dictionary. In every step, we prune away everything but the B most likely hypothesis and save a hypothesis if the `<EOS>`-token is reached. In the end, we select the most likely translation from the set of saved hypotheses.

To improve the performance of their model on a machine translation task, Sutskever et al. reversed the order of their input sequence. This, although it does not change the average distance between corresponding words in the source sentence and the target sentence, leads to a much smaller "minimal time lag" [Hochreiter & Schmidhuber 97b] and appears to improve performance, especially on long sentences.

The fact that such a simple modification in the order of input improves the translation performance indicates that this approach and in particular the bottleneck introduced by the fixed-size representation, is not an optimal solution for the neural machine translation problem. Furthermore, such an approach is not very intuitive since humans do not translate by reading the source sentence, remembering it and writing down the corresponding translation with this in mind. Instead, people look up the source sentence for every word they want to translate. Additionally, they focus on different parts of the source sentence during translation, i.e. they shift their attention.

4.2 Attention-Based NMT

The intuition described above was the inspiration for the attention-based neural machine translation approach that was introduced by Bahdanau et al. [Bahdanau & Cho$^+$ 15] in 2015. The approach is based on three major components: a *bidirectional encoder*, a *decoder* and an *attention mechanism*, which will be described in detail in the following. An overview of the whole architecture is shown in figure 5.2.

Bidirectional Encoder

The bidirectional encoder is simply a bidirectional RNN that encodes the source sentence f_1^J into a vector sequence h_1^J. This is done by applying one RNN $\overrightarrow{g_{\text{enc}}}$ in forward and another RNN $\overleftarrow{g_{\text{enc}}}$ with separate parameters in backward direction to

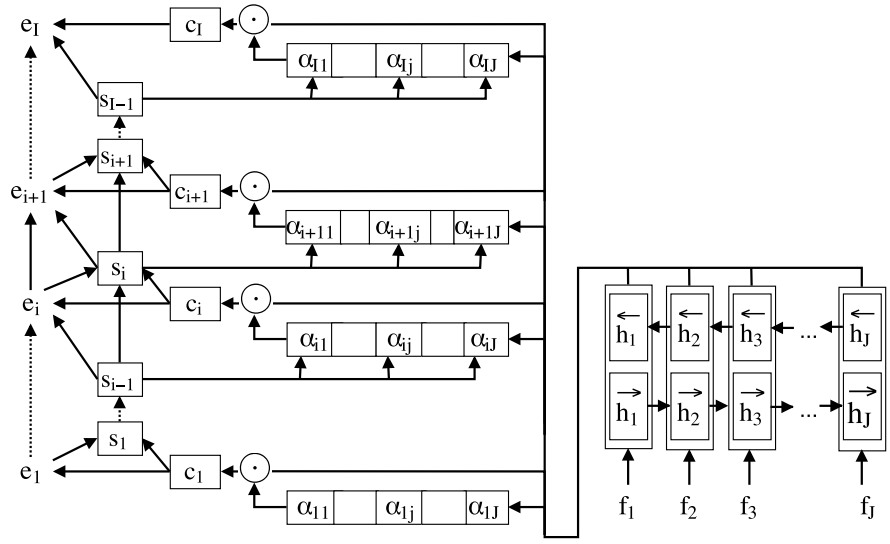

Figure 4.2: The attention-based approach [Bahdanau & Cho$^+$ 15]

the source sentence f_1^J one word at a time, in order to produce a forward representation $\overrightarrow{h}_1^J$ and a backward representation $\overleftarrow{h}_1^J$ from the corresponding RNNs.

$$\overrightarrow{h}_j := \overrightarrow{g_{\text{enc}}}(h_{j-1}, f_j) \text{ for all } j = 1, \ldots, J \tag{4.3}$$

$$\overleftarrow{h}_j := \overleftarrow{g_{\text{enc}}}(h_{j+1}, f_j) \text{ for all } j = 1, \ldots, J \tag{4.4}$$

After this, each sequence element encodes the context of all previous inputs in the corresponding direction. Therefore we can just concatenate both representations to achieve a representation with full context in each position:

$$h_j := [\overrightarrow{h}_j^T; \overleftarrow{h}_j^T]^T \text{ for all } j = 1, \ldots, J \tag{4.5}$$

This representation is kept fixed for the whole translation process and only has to be computed once at the beginning of each translation to save computation time.

Attention Mechanism

The attention mechanism computes a *soft alignment* $\alpha_{i1}, \ldots, \alpha_{iJ}$ in order to produce a *context vector* c_i out of the source representation h_1^J. This is repeated for each decoder step $i = 1, \ldots, I$. To do so, we first compute *energies* $\tilde{\alpha}_{ij}$ for all $j = 1, \ldots, J$, by applying a simple single layer MLP with the previous decoder state s_{i-1} and

the relevant source representation h_j as inputs. We apply a linear transformation on the output of this MLP by multiplying it with v_a^T in order to achieve a scalar valued output as the energy value for each position j.

$$\tilde{\alpha}_{ij} = v_a^T \tanh(W_a s_{i-1} + U_a h_j) \tag{4.6}$$

$W_a \in \mathbb{R}^{n \times n}$ and $U_a \in \mathbb{R}^{n \times 2n}$ are weight matrices of the MLP and $v_a \in \mathbb{R}^n$ is a vector defining the alignment weights through a linear transformation on the output of the MLP. To use these energies in the next computation steps, we need to normalize them by applying a softmax over all $j = 1, \ldots, J$:

$$\alpha_{ij} = \frac{\exp(\tilde{\alpha}_{ij})}{\sum_{k=1}^{J} \exp(\tilde{\alpha}_{ik})} \tag{4.7}$$

This gives us *attention weights*, which are normalized over all source positions $j = 1, \ldots, J$ and therefore can be interpreted as an alignment from source to target side, if we calculate these weights for all target words e_1^I. We can further use these weights to compute the context vector c_i as a weighted average over all source representations:

$$c_i = \sum_{j=1}^{J} \alpha_{ij} h_j \tag{4.8}$$

This can be seen as representation of the aligned context for each specific decoder position $i = 1, \ldots, I$.

Decoder

The decoder works very similarly to the encoder-decoder approach explained in the previous section. It is a simple unidirectional RNN g_{out} that generates the output sentence e_1^I one word after another. Therefore, it takes the previously generated output e_{i-1}, the old decoder state s_{i-1} and the context vector c_i produced by the attention mechanism and the encoder as inputs.

$$p(e_i | e_1^{i-1}, f_1^J) = g_{\text{out}}(s_{i-1}, c_i; e_{i-1}) \tag{4.9}$$

Since the output layer of the decoder is just a soft-max activation over the whole vocabulary, c_i represents f_1^J and e_{i-1} implicitly depends on e_1^{i-2}, we obtain the probability distribution $p(e_i | e_1^{i-1}, f_1^J)$ as output of the decoder. One crucial dependency that is not explicitly part of the modeled distribution is the decoder state s_{i-1}. This hidden state of the decoder is supposed to contain the context within the output sentence and therefore replace an explicit language model. Following this purpose, we compute the new decoder state s_i after we generated the current

output word e_i, since we need to encode e_i in s_i. To do so, we apply an RNN g_{dec} to e_i and the context vector c_i to calculate s_i as the new hidden state.

$$s_i = g_{\text{dec}}(e_i, c_i; s_{i-1}) \tag{4.10}$$

In practice, all the RNNs in decoder and encoder, except for g_{out}, are implemented by LSTMs or variants of it. We will be using GRUs for the implementations in the following chapters.

All in all, the decoder produces the posterior probability of the output word e_i given its predecessors and the input sentence. Internally, this distribution is modeled with the implicit help of an RNN language model, provided by the decoder state, and a soft alignment represented by the attention weights. The network generates output words until it hypothesizes the sentence-end-token and thereby models the probability of the output sentence given the input sentence as a product of the distributions produced in every decoding step:

$$Pr(e_1^I | f_1^J) = \prod_{i=1}^{I} p(e_i | e_1^{i-1}, f_1^J)$$

Since $p(e_i | e_1^{i-1}, f_1^J)$ is directly modeled by the decoder, we obtain a jointly trainable end-to-end approach to directly model the translation probability.

Chapter 5

Advanced Attention Methods

The attention-based approach, as presented in the previous chapter, is the state-of-the-art for NMT, but there is still room for improvement with these models. Perhaps the biggest improvement opportunity lays in the soft alignment generated by the attention mechanism. This theory is backed by the findings of Tu et al. [Tu & Lu$^+$ 16] in their alignment analysis and our alignment analysis in section 6.3. We found that the alignment quality strongly correlates with the translation quality. Furthermore, the alignment quality we get from the baseline NMT system is suboptimal and highly inferior to a statistical alignment e.g. a GIZA-alignment. Additional indicators for a suboptimal alignment by the attention mechanism are over-, respectively under-translations which can be observed in our translation output. These issues show that we have to deal with the same coverage problem in NMT as in statistical machine translation (chapter 2).

One reason that attention-based alignments are still suboptimal could be the fact that they miss the opportunity to take advantage of past alignment information. Including this information has already been proven successful in traditional statistical MT [Koehn & Och$^+$ 03]. In the next sections, we will introduce several approaches that implement these dependencies in different ways. We will also explain new approaches to training NMT models which lead to an improved alignment.

5.1 Related Work

Since its publication, the attention-based approach [Bahdanau & Cho$^+$ 15] and especially the attention mechanism has been the subject of many modifications in an attempt to solve some of the remaining flaws in the NMT approach.

In the following, we will present and discuss a selection of these models. Thereby, we lay our focus on the methods that are related to our work, which we will present later in this chapter.

5.1.1 Coverage Models

An issue commonly addressed in the literature [Tu & Lu$^+$ 16, Mi & Sankaran$^+$ 16, Sankaran & Mi$^+$ 16] on NMT is the coverage problem we mentioned above.

Tu et al. [Tu & Lu$^+$ 16] propose to append annotation vectors to the intermediate representation of NMT models to keep track of the attention history and thereby address the coverage problem. The integration of these annotation vectors β_{i-1j} happens by adding the vector as another input to the computation of the alignment energy (4.6):

$$\tilde{\alpha}_{ij} = v_a^T \tanh(W_a s_{i-1} + U_a h_j + B_a \beta_{i-1j}) \tag{5.1}$$

with $B_a \in \mathbb{R}^{n \times 1}$. They also propose two different approaches to computing the annotation vectors.

One possibility would be the *linguistic coverage model* that calculates β_{ij} as a sum of past alignments at the same encoder step j:

$$\beta_{ij} = \frac{1}{\Phi_j} \sum_{k=1}^{i} \alpha_{kj} \tag{5.2}$$

This sum is normalized by a fertility value for position j that is inspired by the fertility we know from SMT:

$$\Phi_j = M \cdot \sigma_{\text{sigmoid}}(U_\Phi h_j) \tag{5.3}$$

where $U_\Phi \in \mathbb{R}^{1 \times 2n}$ is a trained weight matrix and $M \in \mathbb{R}$ is a predefined constant that specifies the maximum number of target words one source word should be able to produce.

The second possibility to compute the annotation vectors would be to use a *neural network based coverage model*. For this, Tu et al. use an RNN or GRU g_{rec} in order to calculate the annotation vector from the current attention value at the same encoder step α_{ij}, the source representation at the same encoder position h_j, the previous decoder state s_{i-1} and the previous annotation vector for the same encoder time step β_{i-1j}:

$$\beta_{ij} = g_{\text{rec}}(\alpha_{ij}, h_j, s_{i-1}; \beta_{i-1j}) \tag{5.4}$$

In their experiments Tu et al. noticed a significantly improved alignment and translation quality for both coverage models. In direct comparison, the neural network based coverage model outperformed the linguistic coverage model.

Mi et al. [Mi & Sankaran$^+$ 16] proposed a similar approach to the neural network coverage model. They used a GRU g_{rec} to compute the annotation vector from the

current source word e_i, the current attention value in the same position α_{ij} and the previous annotation vector β_{i-1j} as inputs:

$$\beta_{ij} = g_{\text{rec}}(\alpha_{ij}, e_i; \beta_{i-1j}) \tag{5.5}$$

The main difference to Tu et al. [Tu & Lu$^+$ 16], apart from the different inputs for the annotation computation, is the fact that Mi et al. initialize each source word from a specific coverage embedding matrix instead of a uniform distribution. Despite the small differences, Mi et al. achieved similar results to the approach by Tu et al.

5.1.2 Distortion and Fertility Models

Feng et al. [Feng & Liu$^+$ 16] see the lack of distortion and fertility models as the main cause of the alignment problems in NMT, including the coverage problem. To address this, they try to add a distortion model by using the previous context c_{i-1} as additional input for the attention computation:

$$\tilde{\alpha}_{ij} = v_a^T \tanh(W_a s_{i-1} + U_a h_j + V_a c_{i-1}) \tag{5.6}$$

with $V_a \in \mathbb{R}^{n \times 2n}$ as an additional weight matrix. Furthermore, they improve this simple recurrence by adding an explicit hidden state, i.e. by borrowing some parts of an LSTM, to obtain a longer memory.

To add a distortion model to the network, Feng et al. also propose an implicit fertility model by modifying the decoder. Therefore the GRU g_{dec} that calculates the new decoder state s_i (see equation (4.10)) gets extended by an additional *decay gate*:

$$d_i = \sigma(W_{sd} s_{i-1} + W_{ed} e_i + W_{cd} c_i + b_d) \tag{5.7}$$

with weight matrices W_{sd}, W_{ed}, W_{cd} and the bias vector b_d. This newly introduced decay gate is used to update a *decay state*:

$$sd_i = d_i \odot sd_{i-1} \tag{5.8}$$

Finally, the decay state is transformed by a simple MLP and added to the state computation of the standard GRU (cf. equation (3.32)):

$$s_i = [1 - z_i] \odot s_i + z_i \odot \tilde{s}_i + \tanh(V_s sd_i) \tag{5.9}$$

Feng et al. further penalize the decay state by adding a *step-decay* and *left-over* cost to the standard network cost $\mathcal{L}_{\text{CE}}$:

$$\mathcal{L}_{\text{decay}} = \frac{1}{I} \sum_{i=1}^{I} ||sd_i - sd_{i-1}||_2 \tag{5.10}$$

$$\mathcal{L}_{\text{left}} = ||sd_I||_2 \tag{5.11}$$

$$\mathcal{L} = \mathcal{L}_{\text{CE}} - \mathcal{L}_{\text{decay}} - \mathcal{L}_{\text{left}} \tag{5.12}$$

5.1.3 Structural Alignment Biases

Inspired by the traditional statistical alignment methods, Cohn et al. [Cohn & Hoang$^+$ 16] incorporate different methods inspired by IBM and HMM alignment models into the attention computation (4.6) as additional bias terms:

$$\tilde{\alpha}_{ij} = v_a^T \tanh(\ldots + V_{\text{pos}}\psi(i,j,J) + V_{\text{hmm}}\xi_{\text{hmm}}(\alpha_{i-1};j) + V_{\text{fert}}\xi_{\text{fert}}(\alpha_{<i};j)) \quad (5.13)$$

First, they incorporate a *position bias* $\psi(i,j,J)$ which is based on the observation that alignment points tend to occur near the diagonal of the corresponding alignment matrix $A \in I \times J$:

$$\psi(i,j,J) = [\log(1+i), \log(1+j), \log(1+J)]^T \quad (5.14)$$

Inspired by the *Markov condition* that is also utilized by the HMM alignment model [Vogel & Ney$^+$ 96], Cohn et al. incorporate a local alignment history $\xi_{\text{hmm}}(\alpha_{i-1};j)$ for moves inside a window of size $2 \cdot k + 1$:

$$\xi_{\text{hmm}}(\alpha_{i-1};j) = [\alpha_{i-1j-k}, \ldots, \alpha_{i-1j}, \ldots, \alpha_{i-1j+k}]^T \quad (5.15)$$

They incorporate a *local fertility* model $\xi_{\text{fert}}(\alpha_{<i};j)$ in the alignment computation. This calculates the sum over all past alignments, similar to the linguistic coverage model introduced by Tu et al. [Tu & Lu$^+$ 16], but applied for a local window and without any normalization:

$$\xi_{\text{fert}}(\alpha_{<i};j) = [\sum_{i'<i} \alpha_{i'j-k}, \ldots, \sum_{i'<i} \alpha_{i'j}, \ldots, \sum_{i'<i} \alpha_{i'j+k}]^T \quad (5.16)$$

Furthermore, Cohn et al. introduce a more explicit, *global fertility*, method. This method models the summated attention weights $\beta_j = \sum_i \alpha_{ij}$ for each source position over the decoder time, as a Gaussian distribution parametrized by $\mu(h_j)$ and $\sigma^2(h_j)$:

$$p(\beta_j | f_1^J, j) = \mathcal{N}(\mu(h_j), \sigma^2(h_j)) \quad (5.17)$$

This distribution is simply added as an additional term to the network loss function:

$$\mathcal{L} = \mathcal{L}_{\text{CE}} - \sum_j \log p(\beta_j | f_1^J, j) \quad (5.18)$$

Also inspired by traditional, statistical, alignment methods, they proposed a method to symmetrize the training process. Therefore, they jointly train two twinned models on one data set with switched source and target direction, by optimizing a joint loss function:

$$\mathcal{L} = \mathcal{L}_{\text{CE}}(f_1^J | e_1^I) + \mathcal{L}_{\text{CE}}(e_1^I | f_1^J) - B \quad (5.19)$$

In order to force the two models to agree on one alignment, they include an additional *trace bonus*:

$$B = \sum_i \sum_j \alpha_{ji}^{f \leftarrow e} \alpha_{ij}^{f \rightarrow e} \tag{5.20}$$

Combining all these methods, Cohn et al. showed improvements across several challenging language pairs in a low-resource setting.

5.1.4 Temporal Attention

Temporal attention [Sankaran & Mi$^+$ 16] is an approach which introduces an indirect recurrency in the computation of the attention weights, similar to the linguistic coverage model presented above. The temporal attention approach first computes a modulated alignment b_{ij} that is based on the alignment energy $\tilde{\alpha}_{ij}$ and the historical alignment $\sum_{k=1}^{i-1} \exp(\tilde{\alpha}_{kj})$:

$$b_{ij} = \frac{\exp(\tilde{\alpha}_{ij})}{\sum_{k=1}^{i-1} \exp(\tilde{\alpha}_{kj})} \tag{5.21}$$

As for the standard attention weights, the final attention weights need to be normalized over the encoder time:

$$\alpha_{ij} = \frac{b_{ij}}{\sum_{k=1}^{J} b_{ik}} \tag{5.22}$$

This model offers a great conceptual simplicity, since it can be interpreted as a *memory network* [Weston & Chopra$^+$ 14] which memorizes the attention decisions temporally during decoding. Furthermore, this model is also very successful in practice where it outperformed the coverage modeling approaches by Tu et al. [Tu & Lu$^+$ 16] and Mi et al. [Mi & Wang$^+$ 16].

5.1.5 Recurrent Neural Machine Translation

The coverage problem and the commonly occurring errors in long sentence translations were the motivation for the recurrent NMT approach that was introduced by Zhang et al. [Zhang & Xiong$^+$ 16]. They propose a new implicit attention model that is supposed to replace the traditional explicit attention approach that was presented by Bahdanau et al. [Bahdanau & Cho$^+$ 15].

Therefore they use a *contexter*, which is essentially a GRU g_{con} over the source sentence $j = 1, \ldots, J$, which directly computes the context from the source sentence representation h_1^J:

$$c_{ij} = g_{\text{con}}(h_j; c_{ij-1}) \tag{5.23}$$

The dependency on the decoder context is introduced to the contexter by initializing the hidden state of the GRU as the previous decoder state s_{i-1} transformed by an MLP:

$$c_{i0} = \tanh(V s_{i-1} + b_0) \tag{5.24}$$

with the weight parameters $V \in \mathbb{R}^{n \times n}$ and $b_0 \in \mathbb{R}^n$. The context vector can be extracted by using mean-pooling:

$$\tilde{c}_i = \frac{1}{J} \sum_{j=1}^{J} c_{ij} \tag{5.25}$$

or by just copying the final hidden state of the contexter:

$$c_i = c_{iJ} \tag{5.26}$$

Therefore, we generate the context vector from the same inputs as the traditional, explicit, attention approach. This approach produced significant improvements over standard attention-based NMT, but even though the dimension of the context vector was halved in contrast to standard NMT, it took about 3.5-times as long to train this model.

5.2 Alignment Feedback

As mentioned earlier, the standard attention-mechanism [Bahdanau & Cho$^+$ 15] does not take advantage of any prior alignment information. Some of the methods we presented in the previous sections try to include this information in their computations, which results in significant improvements in all cases. Following their example, we also use past alignment information in our computations. Therefore, similar to the coverage models [Tu & Lu$^+$ 16, Mi & Wang$^+$ 16], we include the alignment context in the energy computation (4.6), by introducing a feedback component γ_{i-1j}:

$$\tilde{\alpha}_{ij} = v_a^T \tanh(W_a s_{i-1} + U_a h_j + V_a \gamma_{i-1j}) \tag{5.27}$$

In the following, we will present two possible approaches to compute this feedback component from the attention weights α_{i-1j} generated in the previous decoder time step.

5.2.1 Convolutional Feedback

The standard attention mechanism, as it was originally introduced by Bahdanau et al. [Bahdanau & Cho$^+$ 15], implements a content-based attention. That means

that the attention has no direct dependency on the position it is computed for. Chorowski et al. argue that this independence assumption does not hold for monotonous alignments, as they are given e.g. in speech recognition. Even though the alignments in machine translation are not generally monotonous, we still have local monotonicity in most cases.

Therefore, we decided to implement the hybrid attention mechanism that Chorowski et al. [Chorowski & Bahdanau$^+$ 15] proposed to adapt the attention-based approach to speech recognition. For this, we define the feedback component γ_i as a one-dimensional convolutional operation over the attention weights α_i:

$$\gamma_i = G * \alpha_i \tag{5.28}$$

where $G \in \mathbb{R}^{M \times 2k+1}$. This means that we extract M vectors, where for each vector we applied a different filter (weight matrix) G_m. Every filter G_m extracts information from a window of size $2k+1$ that is centered around position j:

$$\gamma_{m,ij} = \sum_{l=j-k}^{j+k} G_{m,j-l} \cdot \alpha_{il} \quad \text{for all } m = 1, \ldots, M \tag{5.29}$$

When added to the attention computation, as specified in equation (5.27), the network can decide in every step how much information it wants to extract from the original position-independent information or from the introduced position-dependent alignment feedback information.

5.2.2 Bidirectional RNN Feedback

The experimental results for convolutional feedback (cf. chapter 6) showed that using only one filter with a big filter size gives results that were similar to models with more filters and a smaller window size. The bigger this window becomes, the more positional information is lost until we fall back to a content-based attention mechanism. Since a large window sizes showed very promising results, we decided to ignore most positional dependencies and use the whole context for the attention weights.

The idea is to apply a bidirectional RNN g_{rec} that encodes the whole alignment context for each position $j = 1, \ldots, J$. Therefore, we apply one RNN for each direction to the scalar-valued attention weights of decoder time step i:

$$\overrightarrow{\gamma}_{ij} = \overrightarrow{g_{\text{rec}}}(\alpha_{ij}; \overrightarrow{\gamma}_{ij-1}) \tag{5.30}$$

$$\overleftarrow{\gamma}_{ij} = \overrightarrow{g_{\text{rec}}}(\alpha_{ij}; \overleftarrow{\gamma}_{ij+1}) \tag{5.31}$$

The resulting outputs get concatenated to form the feedback component that should encode the whole alignment context of decoding time step i:

$$\gamma_{ij} = [\overrightarrow{\gamma}_{ij}^T; \overleftarrow{\gamma}_{ij}^T]^T \tag{5.32}$$

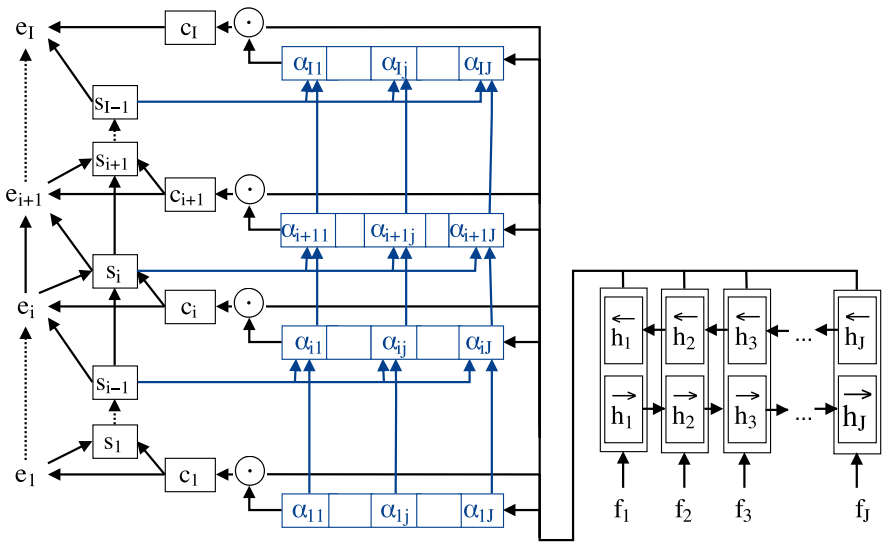

Figure 5.1: The recurrent attention approach.

5.3 Recurrent Attention Computation

In the previous section, we have presented methods to add information about the previous alignment to the standard attention computation. To be precise, we added the feedback component γ_{i-1j} as an additional input to the MLP that computes the energies. But even though, for every $\tilde{\alpha}_{ij}$, we add a representation of the alignments from time step $i-1$ around position j, we still not consider the unmodified context $\tilde{\alpha}_{i-1j}$. If we replace the energy computer MLP by g_{rec}, an RNN or even an LSTM, we have the full context $\tilde{\alpha}_1^{i-1}{}_j$ included in the energy computation for every position j:

$$\tilde{\alpha}_{ij} = v_a^T \cdot g_{\text{rec}}(s_{i-1}, h_j; \tilde{\alpha}_{i-1j}) \tag{5.33}$$

Since the context of the same position only contains the information about whether the same source position has been aligned before or not, it makes sense to combine this recurrent attention computation with the alignment feedback approach from the previous section in order to include context around the position j:

$$\tilde{\alpha}_{ij} = v_a^T \cdot g_{\text{rec}}(s_{i-1}, h_j, \gamma_{i-1j}; \tilde{\alpha}_{i-1j}) \tag{5.34}$$

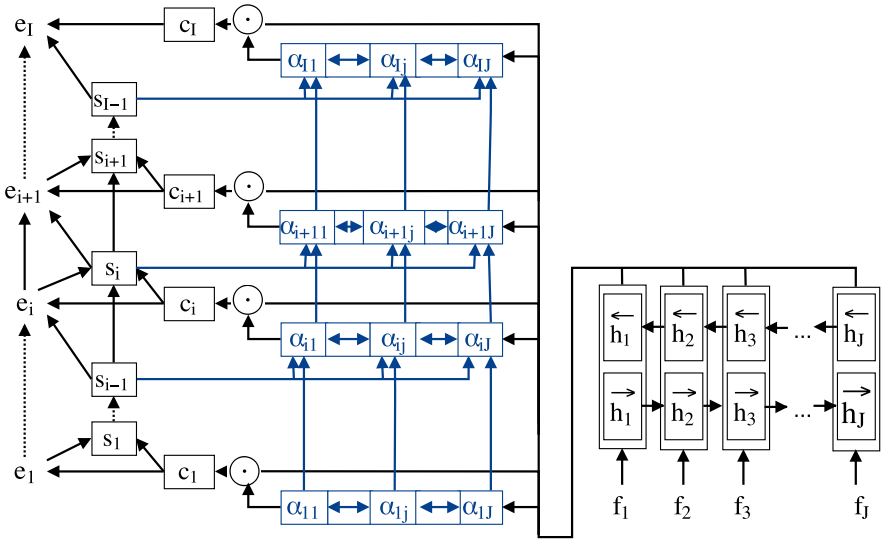

Figure 5.2: The multidimensional attention approach.

5.3.1 Multidimensional Attention Computation

The recurrent attention computation includes, when implemented using an LSTM or GRU, the information of all previous decoder time steps at the same position ${\tilde{\alpha}_1^{i-1}}_j$. What may be even more important than this information is the context over the encoding steps $\tilde{\alpha}_{i1}^{j-1}$ and $\tilde{\alpha}_{ij+1}^{J}$. Including this information to the computation of every $\tilde{\alpha}_{ij}$ would make the alignments interdependent for the same decoder time step i, which was already succesfully applied in the recurrent NMT approach (section 5.1.5). The intuition behind this is that human translators would often look at multiple source words when they add another word to the translated sentence. Also an additonal recurrent connection over the decoder time could make sense here, since word groups on the encoder side often get translated to one word group on the decoder side. This concept was already successfully applied to statistical MT with the phrase based approach (section 2.4).

To include both the preceding context over decoder time $\tilde{\alpha}_{1j}^{i-1}$ and the full attention context over encoder time $\tilde{\alpha}_{i1}^{j-1}$ and $\tilde{\alpha}_{ij+1}^{J}$, implement a *two-dimensional RNN* [Graves & Fernández$^+$ 07] which is bidirectional over the encoder time. Following Graves et al. [Graves & Fernández$^+$ 07], we have two opposite RNNs $\overrightarrow{g_{\text{md}}}$ and $\overleftarrow{g_{\text{md}}}$ (cf. figure 5.3). $\overrightarrow{g_{\text{md}}}$ takes the alignment from the previous decoder step

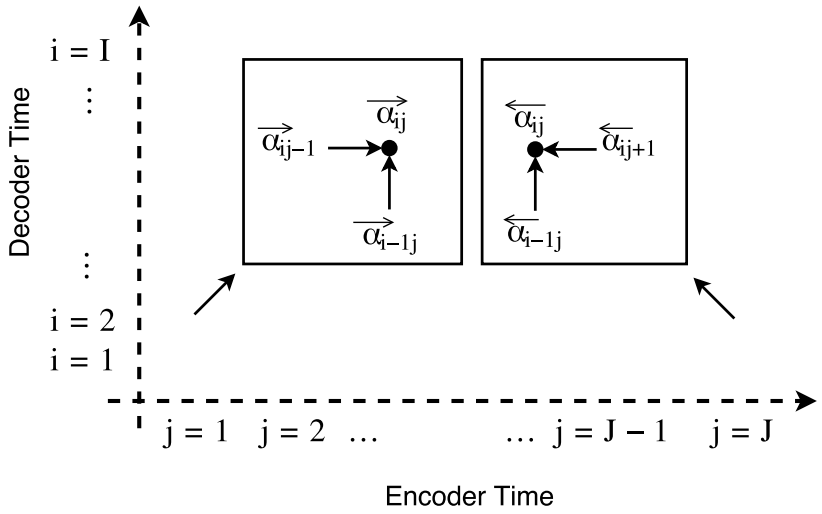

Figure 5.3: Concept of the MDRNN for attention

for the same encoder position $\overrightarrow{\alpha}_{i-1j}$ and the alignment from the same decoder step and the previous encoder step $\overrightarrow{\alpha}_{ij-1}$ as context:

$$\overrightarrow{\alpha}_{ij} = \overrightarrow{g_{\text{md}}}(s_{i-1}, h_j; \overrightarrow{\alpha}_{i-1j}, \overrightarrow{\alpha}_{ij-1}) \tag{5.35}$$

The second RNN $\overleftarrow{g_{\text{md}}}$ is mirrored over the encoder time, i.e. it has the same context over the decoder time $\overleftarrow{\alpha}_{i-1j}$, but the context from the next encoder time step $\overleftarrow{\alpha}_{ij+1}$:

$$\overleftarrow{\alpha}_{ij} = \overleftarrow{g_{\text{md}}}(s_{i-1}, h_j; \overleftarrow{\alpha}_{i-1j}, \overleftarrow{\alpha}_{ij+1}) \tag{5.36}$$

As for any one-dimensional bidirectional RNN, we simply concatenate both directions of the attention computation. Then we still need to apply a linear transformation, similar to the non-recurrent energy computation:

$$\tilde{\alpha}_{ij} = v_a^T \cdot [\overrightarrow{\alpha_{ij}}^T; \overleftarrow{\alpha_{ij}}^T]^T \tag{5.37}$$

with $v_a \in \mathbb{R}^{2 \cdot n}$ to obtain a scalar value for the following computations which may remain unchanged to the standard attention based approach.

Multidimesional Gated Recurrent Unit

For an efficient and stable implementation of a multidimensional RNN, Graves et al. [Graves & Fernández$^+$ 07] also modified the standard LSTM structure to perform its computations w.r.t. cell states from multiple directions, by introducing a

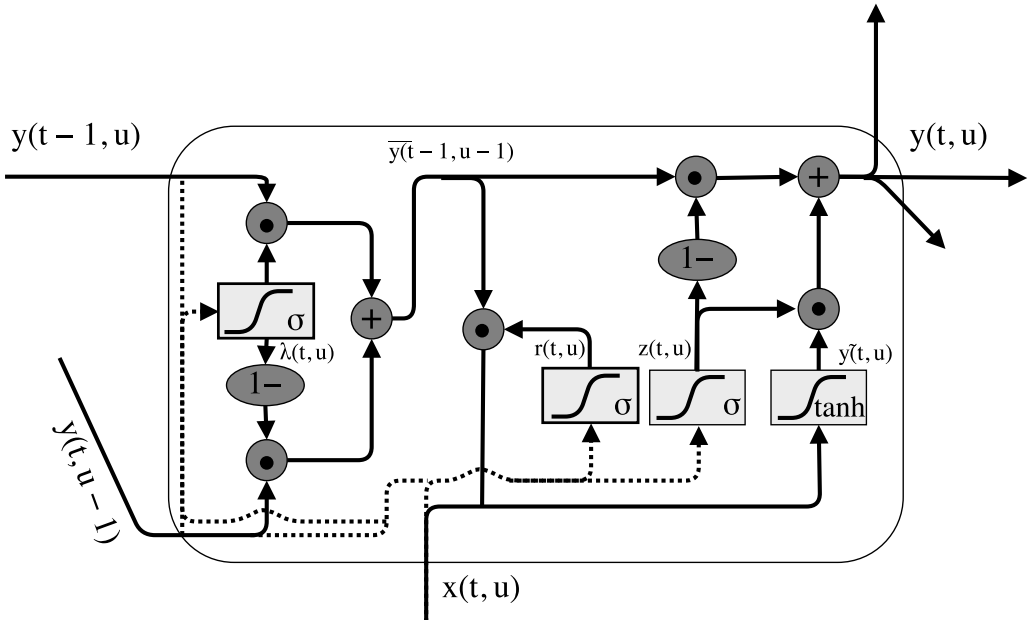

Figure 5.4: Architecture of an MDGRU

multidimensional LSTM (MDLSTM). Unfortunately, the standard MDLSTM still suffers from the same problem of its one-dimensional counterpart, namely the increased computation time due to the additional parameters. Since this problem is only aggravated with every additional context dimension we consider, we propose a variation of the GRU architecture to implement a multidimensional GRU (MDGRU) that is inspired by the MDLSTM architecture.

For simplicity, we will only consider the two-dimensional case for the MDGRU formulation. To obtain a stable architecture, we follow the example of Leifert et al. [Leifert & Strauß$^+$ 14] by adding the contexts $y(t-1,u)$ and $y(t,u-1)$ from both directions weighted through an additional lambda-gate $\lambda(t,u)$ in order to combine them to a single context representation:

$$\overline{y}(t-1,u-1) = \lambda(t,u) \odot y(t-1,u) + [1-\lambda(t,u)] \odot y(t,u-1) \tag{5.38}$$

Given this combined context, we can easily modify the standard GRU formulas (3.32) to (3.35) by using an additional timestep parameter u. This way, we continue to use the update gate $z(t,u)$ to compute the new output $y(t,u)$ as a linear

interpolation between the combined context $\overline{y}(t-1,u-1)$ and the candidate activation $\tilde{y}(t,u)$:

$$y(t,u) = [1 - z(t,u)] \odot \overline{y}(t-1,u-1) + z(t,u) \odot \tilde{y}(t,u) \tag{5.39}$$

The candidate activation similarly depends on the current input $x(t,u)$ and again the combined context $\overline{y}(t-1,u-1)$, filtered by the reset gate $r(t,u)$:

$$\tilde{y}(t,u) = \sigma_{\tanh}(W_{xy}x(t,u) + W_{yy}[r(t,u) \odot \overline{y}(t-1,u-1)] + b_y) \tag{5.40}$$

The update and reset gates as well as the introduced lambda gate are also computed similarly to the one-dimensional GRU. The difference is that we need to consider the contexts from both directions in the gate computation. Therefore, we need separate weight matrices W_{y*} and U_{y*} for $* \in \{r, z, \lambda\}$ to add both contexts into the gate computation.

$$r(t,u) = \sigma(W_{xr}x(t,u) + W_{yr}y(t-1,u) + U_{yr}y(t,u-1) + b_r) \tag{5.41}$$
$$z(t,u) = \sigma(W_{xz}x(t,u) + W_{yz}y(t-1,u) + U_{yz}y(t,u-1) + b_z) \tag{5.42}$$
$$\lambda(t,u) = \sigma(W_{x\lambda}x(t,u) + W_{y\lambda}y(t-1,u) + U_{y\lambda}y(t,u-1) + b_\lambda) \tag{5.43}$$

The rest of the parameters can be adapted from the one-dimensional case. Thus, the parameter reduction we know from the one-dimensional GRU is even reinforced if you compare the two-dimensional GRU to the two-dimensional LSTM. And in a preliminary experiment, both MDLSTM and MDGRU achieved similar results, while the MDGRU speeds up the computation time significantly.

5.4 Guided Alignment Training

The concept of using an implicit soft alignment model, integrated in the network through the attention mechanism, has many advantages. The greatest advantage is that we can train the model jointly with the whole network and optimize with respect to the network cost function $\mathcal{L}$ defined on the output level. This decoder cost function is chosen, as proposed by Bahdanau and others [Bahdanau & Cho$^+$ 15], as the conditional log-likelihood of the target sentence e_1^I given f_1^J, i.e. the cross-entropy:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N}\sum_n \log p_\theta(e_{n,1}^{\ I_n} | f_{n,1}^{\ J_n}) \tag{5.44}$$

Where N is the size of the subset of the training data for which we want to compute the loss and p_θ is the probability distribution modeled by the neural network with parameter set θ.

Even though this approach trains the attention mechanism to generate a soft alignment that leads to minimizing the decoder cost function, this might also be a disadvantage. The attention-mechanism might still not be able to generate an optimal alignment that could lead to better results with respect to our ultimate evaluation criterion which is different from the standard decoder cost. This might also be due to the fact that it is hard to propagate the part of the error, that is caused by the attention mechanism, back through the whole network to the attention layer. Thus, the training might get caught in a local minimum.

To prevent this problem, Chen et al. [Chen & Matusov$^+$ 16] and similarly Mi et al. [Mi & Wang$^+$ 16] introduced a second cost function that gives feedback explicitly for the components of the attention mechanism. They compute this second loss function as the cross-entropy between the soft alignment α_{ij} extracted from the attention mechanism and a given target alignment A_{ij}:

$$\mathcal{L}_{\text{align}}(A, \alpha) := -\frac{1}{N} \sum_{n} \sum_{i=1}^{I_n} \sum_{j=1}^{J_n} A_{n,ij} \log \alpha_{n,ij} \tag{5.45}$$

To integrate this additional error measure into the traditional training process, we define a new network loss function as the weighted sum of the standard decoder cost function (5.44) and the introduced alignment cost function (5.45):

$$\mathcal{L} := \lambda_{\text{CE}} \cdot \mathcal{L}_{\text{CE}} + \lambda_{\text{align}} \cdot \mathcal{L}_{\text{align}} \tag{5.46}$$

with the weight factors $\lambda_{\text{CE}} \in \mathbb{R}$ and $\lambda_{\text{align}} \in \mathbb{R}$ that need to be optimized separately.

The effect of the alignment cost function highly depends on the quality of the utilized target alignment A_{ij}. Since it is too expensive to create a perfect alignment for the whole training set by hand, we use the Viterbi alignment generated by GIZA++ as a synthetic target alignment. Our alignment experiments (section 6.3) showed that the GIZA-alignment is still much better than the baseline soft alignment we extracted from the standard attention mechanism. Therefore, we can confidently expect that the error measure we obtain from $\mathcal{L}_{\text{align}}$, with the GIZA-alignment as the target, will give a good estimate of the error we could expect compared to manually produced alignments. To use the hard alignments $\tilde{A}_{ij}$ produced by GIZA++, we need to normalize them over the source direction:

$$A_{ij} = \frac{\tilde{A}_{ij}}{\sum_{k=1}^{J} \tilde{A}_{ik}} \quad \text{for all } i = 1, \ldots, I \tag{5.47}$$

This way we obtain a synthetic soft alignment that fulfills the constraints imposed by the cross-entropy cost function.

5.5 Alignment Foresight

The problem with the guided alignment training described in the previous section is the fact that we still rely on hard alignments generated by GIZA++. This is bad for several reasons: First of all, GIZA++ is designed to produce alignments which are restricted to hard decisions between zero and one for each alignment position. This is in contrast to the soft alignments generated by an attention mechanism, where each position gives a value between zero and one, which can be interpreted as a probability that states how likely this alignment should be considered. Furthermore, GIZA++ alignments are not optimal [Och & Ney 03] and therefore it may be that using these alignments as additional synthetic training data, may limit our system and prohibit significant improvements of the alignment quality in some cases.

To replace GIZA++ alignments for guided alignment training and to analyze the alignment capabilities of attention based systems, we use the knowledge of the target sentence e_1^I we have in training to improve the attention. For this, we introduce the target word of the current decoding step e_i as additional input for the alignment energy computation:

$$\tilde{\alpha}_{ij} = v_a^T \tanh(W_a s_{i-1} + U_a h_j + V_a \tilde{e}_i)$$

where $\tilde{e}_i$ is the output of a projection layer, which has to be separate from the one used to embed the source words. The idea behind this method we call *alignment foresight* is that the network should be able to use this perfect knowledge of the target word e_i to select the right source words that need to be aligned in order to generate e_i.

Chapter 6

Experiments

In order to measure the effect of the advanced attention methods described in the previous chapter, experiments were conducted on a German to English and an English to Romanian translation task. The setup, expiration and the results of these translation experiments as well as an additional alignment analysis are described and discussed in detail in this chapter.

6.1 Experiment Setup

The experiments in this chapter were are all conducted with models that are based on the standard attention-based NMT system, which is configurated according to the settings we specify in the next section. Additionally, we also describe the training setup and corpus statistics for the different tasks we use for the evaluation.

6.1.1 System Configuration

The models we use for all experiments that we describe in the next sections are based on the attention-based NMT approach by Bahdanau et al. (cf. section 4.2). In this model, we use a word-embedding size of 620 for the projection layer. The decoder as well as both directed RNNs of the bi-directional encoder are composed of 1000 GRU nodes. Moreover, the attention mechanism also has an internal dimension of 1000 that is used for the alignment computation. For decoding, we use a beam-size B of 12. We implemented the NMT system based on the *Blocks* framework [Van Merriënboer & Bahdanau$^+$ 15] and the deep-learning library *Theano* [Bergstra & Breuleux$^+$ 10].

The vocabularies for source and target language are created from the 30K most frequent words in the corresponding sentences in the training corpus. Such a limited vocabulary size can lead to a relatively high out-of-vocabulary (OOV) rate which normally leads to many translation errors. To reduce such translation errors, we apply a direct mapping from source to target side if unknown tokens occur on the target side after translation. Therefore, we replace the unknown token by the word in the source sentence where the strongest attention weight points to.

Additionally, to reduce the OOV-rate, we apply *tokenization* to training and translation data, that means we separate the punctuation marks from the surrounding word. This way, the system interprets the punctuation marks as normal tokens that are part of the input and can also be hypothesized. We tokenize the data in a preprocessing step and undo tokenization in a final postprocessing step. Similarly to unknown replacement, we also apply *categorization*, i.e. we mark numbers, hours and dates as special tokens that just get transferred from source to target side directly if the system hypothesizes the corresponding token.

6.1.2 Training

For the experiments we conducted to evaluate the translation quality of our models, we used two different data sets to train our models for different tasks. The detailed statistics of all data sets we use for evaluation can be found in the appendix (A.2).

IWSLT2013: German - English

The largest task is the *IWSLT2013* [Cettolo & Girardi$^+$ 12] German to English task which is composed of 4.3 million bilingual sentence pairs with a subset of 138K sentence-pairs of data from the target domain. As a first preprocessing step, we always apply word-reordering on the German sentences of this data set. For our experiments, we always train our models on the full data set with in-domain data included twice to increase domain adaptation. Therefore, we have a training set of 4.4 million sentence pairs and 111 million running words in both German and English.

On this large data set, we train our models for 500K iterations, which takes about a week for our baseline model when trained on a Nvidia GTX 980 GPU. During training, we evaluate the translation performance a `development` set, containing 887 sentences, after each 10K iterations. After training, the model from the iteration that resulted in the best BLEU-score on the development set is selected for the final evaluation on the development, `eval` and `test` sets.

WMT2016: English - Romanian

The second data set for our experiments is from the *WMT2016*[1] English to Romanian task, and has much fewer data compared to IWSLT2013. We train on a set of 211K bilingual sentence pairs which consist of 5.0 million and 5.3 million running word in English respectively Romanian. Due to the reduced corpus size, we only train up to 300K iterations. The training is monitored by evaluating the translation quality after every 10K iterations on the `newsdev2016_1` set. Analogously to the

[1] `http://www.statmt.org/wmt16/translation-task.html`

IWSLT2013 experiments, we choose the best model among these development evaluations. The final evaluation is performed on the `newsdev2016_1`, `newsdev2016_2` and `newstest2016` sets.

Europarl: German - English

To evaluate the alignment quality of our models, we use a set of 504 bilingual sentence pairs that were extracted from the *Europarl* [Koehn 05] German to English task and manually aligned by human annotators. We use this `align-test` set to evaluate the alignment quality of the models we trained on the IWSLT data which should lead to a good estimation of the alignment quality since the Europarl data is included in the training data of the IWSLT2013 task.

However, if we solely want to analyze the alignment quality and especially if we want to compare our results with the alignments generated by GIZA++, we reduce the training data to the Europarl training set. This training data consists of 1.2 million bilingual sentences of 32/34 million running words in German/English. The training takes 250K iterations, and the model is selected by translation quality on the `developement` set from the IWSLT2013 task, on which it is evaluated after every 10K iterations.

For the comparisons with GIZA++ that we conduct in the next section, we trained the alignments on the Europarl data with five iterations of the IBM-1 model, followed by five iterations of the HMM model and five final iterations of the IBM-4 model.

6.2 Evaluation Methods

To reliably and deterministically evaluate and compare our models, an evaluation method that fits the requirements is needed. To evaluate the translation quality, we chose the commonly used BLEU and TER measures for the experiments in this thesis. Since we also analyze alignments, it is necessary to use a properly defined evaluation method for this task. We present the AER, F-Measure and SAER methods in this chapter. Which method is used for our evaluations is finally determined through comparisons in multiple preliminary experiments.

6.2.1 BLEU

The most common evaluation measure for machine translation models is the *bilingual evaluation understudy (BLEU)* algorithm [Papineni & Roukos$^+$ 02]. This algorithm was introduced to compare a candidate translation e_1^I, which usually consists of the whole translated test set, against multiple reference translations. For this thesis, we only consider the case of a single reference translation $\hat{e}_1^{\hat{I}}$. To

compare e_1^I to $\hat{e}_1^{\hat{I}}$, we use the modified n-gram precision. For every n-gram w_1^n in the candidate we calculate the maximum of the total count of w_1^n in e_1^I and the total count of w_1^n in $\hat{e}_1^{\hat{I}}$. This clipped word count divided by the total number of n-grams in the candidate translation e_1^I and summed for each w_1^n in e_1^I produces the modified n-gram precision score:

$$\mathrm{Prec}_n(e_1^I, \hat{e}_1^{\hat{I}}) = \frac{\sum_{w_1^n} \min\{c(w_1^n|e_1^I), c(w_1^n|\hat{e}_1^{\hat{I}})\}}{\sum_{w_1^n} c(w_1^n|e_1^I)} \tag{6.1}$$

Additionally, we need a brevity penalty in order to prevent very short candidates from receiving an unjustifiable high score.

$$\mathrm{BP}(I, \hat{I}) = \begin{cases} 1 & \text{if } I \geq \hat{I} \\ \exp(1 - \frac{I}{\hat{I}}) & \text{if } I < \hat{I} \end{cases} \tag{6.2}$$

The BLEU score is then calculated as the geometric average of the modified n-gram precision Prec_n multiplied with the brevity penalty BP:

$$\mathrm{BLEU}(e_1^I, \hat{e}_1^{\hat{I}}) = \mathrm{BP}(I, \hat{I}) \cdot \prod_{m=1}^{n} \mathrm{Prec}_n(e_1^I, \hat{e}_1^{\hat{I}})^{\frac{1}{n}} \tag{6.3}$$

with $n = 4$ for our experiments.

The BLEU-score has been proven to produce scores that correlate well with human evaluation [Papineni & Roukos$^+$ 02] and is, therefore, very popular for evaluation of machine translation systems.

6.2.2 TER

In addition to the BLEU-score, we use the *translation edit rate (TER)* [Snover & Dorr$^+$ 06] to evaluate the translation quality of our system. The TER is defined as the number of edits that is needed to transform a candidate translation e_1^I so that it exactly matches the reference translation $\hat{e}_1^{\hat{I}}$. This value is divided by the average number of words in the reference translations or, in our case, the number of words in the single reference translation:

$$\mathrm{TER}(e_1^I, \hat{e}_1^{\hat{I}}) = \frac{\#\text{edits}}{\hat{I}} \tag{6.4}$$

Edit operations include insertions, deletions, substitutions and, in contrast to word error rate, also shifts.

6.2.3 Alignment Evaluation Methods

Additionally, to analyzing the translation quality of an MT system, it can also be helpful to evaluate the quality of the alignments the system generates when it produces these translations.

To assess the alignments, we need a manually aligned set of bilingual sentence pairs. That means we need a set $\hat{A}$ of alignment points which were produced by humans for each sentence pair $(f_1^J, \hat{e}_1^{\hat{I}})$. Since the alignments produced by humans are not deterministic (i.e. two different experts may very well produce different alignments for the same data), Och and Ney [Och & Ney 03] introduced the usage of "sure" and "possible" alignments. With this convention, we obtain one set S with the alignment points the human annotator considers as unambiguous, and another set P for the ambiguous alignments. Additionally, all alignment points in S need to be included in P, which results in $S \subseteq P$. Also P contains all alignment points the human annotator deemed ambiguous or unambiguous, i.e. all points in $\hat{A}$.

The soft alignment we generate with the attention mechanism can be used to create a set of alignment points A that represents the hard alignment representation of the attention values. To use as much information as possible, we extract two hard alignments $A_{i\to j}$ and $A_{i\leftarrow j}$ from the soft alignment α:

$$(i,j) \in A_{i\to j} \text{ iff } \alpha_{ij} \geq \alpha_{ik} \text{ for all } k = 1,\ldots,J \tag{6.5}$$

$$(i,j) \in A_{i\leftarrow j} \text{ iff } \alpha_{ij} \geq \alpha_{hj} \text{ for all } h = 1,\ldots,I \tag{6.6}$$

These two alignments are then merged by applying the heuristic presented by Och and Ney [Och & Ney 03] as we explained it in section 2.2.2. With the resulting alignment and a corresponding manual alignment, we can simply apply the traditional alignment evaluation methods AER and F-Measure that we describe in the following.

AER

Given a manual alignment (S, P) in the way we described above, Och and Ney [Och & Ney 03] defined the commonly used *alignment error rate (AER)*, which is derived from the well-known *F-Measure*:

$$\mathrm{AER}(S,P;A) = 1 - \frac{|A \cap S| + |A \cap P|}{|A| + |S|} \tag{6.7}$$

F-Measure

As Fraser and Marcu mentioned in their work [Fraser & Marcu 07], AER lacks the important property to penalize unbalanced precision and recall values. In order to correct this, they introduce a modified version of the standard F-Measure formula:

$$\text{F-Measure}(A, P, S, \varphi) = \frac{1}{\frac{\varphi}{\text{Precision}(A,P)} + \frac{1-\varphi}{\text{Recall}(A,S)}} \tag{6.8}$$

with the appropriately redefined Precision and Recall measures taken from Och and Ney [Och & Ney 03]:

$$\text{Precision}(A, P) = \frac{|A \cap P|}{|A|} \tag{6.9}$$

$$\text{Recall}(A, S) = \frac{|A \cap S|}{|S|} \tag{6.10}$$

This includes a parameter φ which can be used to specify the trade-off between precision and recall.

SAER

Since the evaluation measures described above are all designed for hard alignments produced by traditional statistical alignment methods, they may not be ideal to evaluate soft alignments generated by attention-based NMT models. With this in mind, Tu et al. [Tu & Lu$^+$ 16] introduced the *soft alignment error rate (SAER)*:

$$\text{SAER}(M_S, M_P; M_A) = 1 - \frac{|M_A \odot M_S| + |M_A \odot M_P|}{|M_A| + |M_S|} \tag{6.11}$$

where $\odot$ is an elementwise matrix multiplication and $|M|$ gives the sum of all elements in matrix M. The sets of reference alignment points P and S are used to generate the matrices M_P and M_S with the same shape as M_A where $M_P(i, j)$ is set to one iff $(i, j) \in P$ and similarly for M_S and S.

6.3 Alignment Evaluation

Since Bahdanau et al. introduced the attention based NMT approach in 2015 [Bahdanau & Cho$^+$ 15], there have been numerous attempts to improve the translation quality of the system by modifying the attention mechanism. This method builds on the intuition that an improved attention mechanism would lead to a better soft alignment which should help the decoder in translation. However, even though most of the attempts that tried to improve the attention mechanism reported great

improvements in translation quality, to the best of our knowledge, there are only three publications [Tu & Lu$^+$ 16, Mi & Wang$^+$ 16, Sankaran & Mi$^+$ 16] that also evaluated the alignment quality itself.

Nevertheless, there are still many aspects of attention-based NMT alignments that remain uncertain and which we will investigate in the next sections.

6.3.1 Alignment Analysis

One open question in the field of attention-based NMT is how important the attention mechanism is for the translation quality. This is particularly questionable since the encoder-decoder approach generated translations that were competitive to state-of-the-art SMT approaches without any explicit alignment method. Therefore, in the following, we aim to determine the impact of the soft alignments on the resulting translation and on the training process itself. We further want to assess how meaningful the traditional alignment evaluation methods AER and F-Measure and newly introduced SAER are w.r.t. soft alignments.

To investigate this, we conducted an experiment where we trained a baseline attention-based NMT model (i.e. without any additional features) on the IWSLT data set. With this model as our "optimal" soft alignment, we tried to create increasingly poor alignments by adding noise to the model parameters that are involved in the alignment process. The noise is applied by adding random values that are taken from a normal distribution around 0 with increasing standard deviation to add more noise. We increased the standard deviation in steps of 0.01 from 0.0 to 0.15 and calculated the BLEU score and all important alignment quality measures for each model we produced. The same experiment was repeated with a model that was trained on the Europarl data set with very similar results. The graph resulting from the experiments with both training sets can be seen in figure 6.1.

Both experiments showed that the translation quality and also the alignment quality are robust to noise up for a standard deviation of up to 0.04. After this point, the accuracy of alignment and translation degrades rapidly. We also calculated the Pearson coefficient for all alignment quality measures w.r.t. BLEU score. The resulting coefficients were greater than 0.988 for all measures, where a value of 1.0 or -1.0 would indicate a perfect correlation. This means that the alignment quality highly correlates with translation quality when we purposely disturb the attention computation with noise.

Therefore we can say that AER, F-Measure with every $\varphi = 0.0, 0.1, \ldots, 1.0$ and especially the SAER with the highest Pearson coefficient, can serve as a good indicator for the alignment quality of NMT models. However, as we see later, SAER is problematic for comparisons between hard and soft alignments. Nevertheless, we make the reasonable assumption that the alignment error measures indicate soft alignment quality, as well as they do for hard alignments [Fraser & Marcu 07].

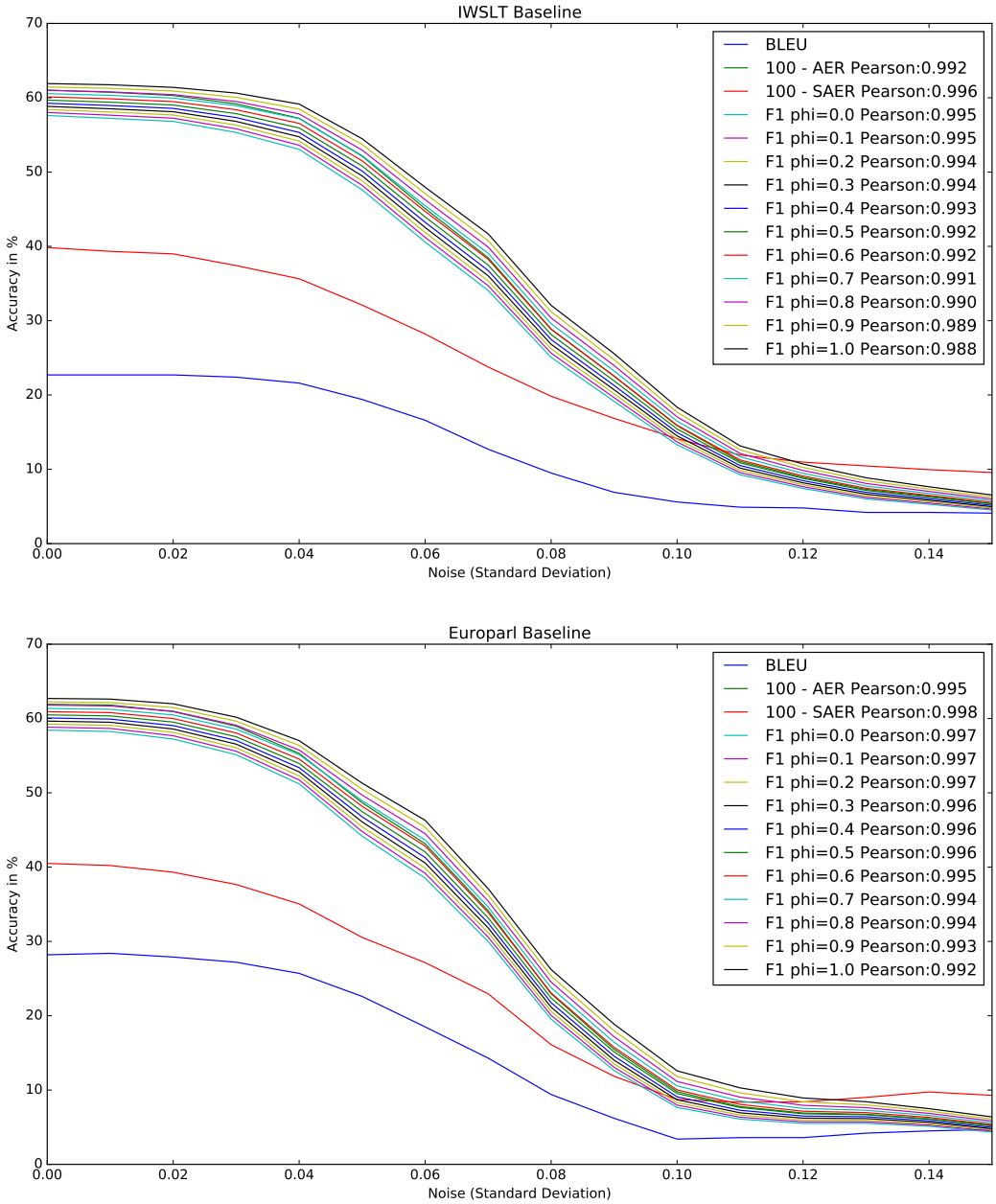

Figure 6.1: Artificially disturbed attention mechanism of a baseline NMT model. Drawn are the alignment quality measures and the BLEU-score over the increasing amount of noise. The Pearson coefficient w.r.t. the BLEU-score for each alignment evaluation method is noted in the legend.

Making this assumption, we can conclude from the observed results that the quality of the soft alignment that is produced by the attention mechanism has a great impact on the quality of the generated translation.

All these conclusions also fit the improvements we have seen from simple modifications in the attention mechanism [Tu & Lu$^+$ 16, Mi & Sankaran$^+$ 16, Sankaran & Mi$^+$ 16, Feng & Liu$^+$ 16, Cohn & Hoang$^+$ 16]. These results and the conclusions we drew from the experiment that is described above, motivated the main part of this work, which was the attempt to improve the alignment and translation quality by advancing the attention mechanism.

6.3.2 Alignment Development during Training

Another aspect of alignment we investigated as part of our work was the development during training. This was motivated by the fact that we believe that the alignment information, which gets compressed into scalars when it is encoded in the attention weights, may make it hard to propagate the error information back to the erroneous parts of the attention mechanism. To examine this, we recorded the BLEU-score on the `developement` set and the SAER on the `alignment-test` set during training after every 10K iterations. This was done for the baseline NMT model trained on IWSLT2013 once using regular training and once using guided alignment training.

The results in figure 6.2 clearly show that guided alignment training results in a slightly higher BLEU-score and significantly decreased SAER during the whole training process. The figure also shows that the beginning of the guided alignment training is much more stable, maybe due to the smoother development we observe in alignment quality.

We conclude that the network benefits from an additional objective that serves as a direct error signal for the attention mechanism during training.

6.3.3 Alignment Foresight

Finally in our alignment analysis, we wanted to explore the limits of attention-based soft alignments. To do so, we attempted in creating an optimal soft alignment using alignment foresight (section 5.5). This was motivated by the poor AER values we achieved with the baseline attention-based NMT model in comparison to GIZA++.

When we naively applied the alignment foresight approach to the network, we ran into the problem that the network does not learn any useful alignment but instead uses the attention weights to encode the target word e_i. This behavior is visible in the heatmap plots of the attention weight matrix as it is shown in figure 6.3. To prevent this behavior, we tried to cancel out the encoding and enforce a semi-hard alignment by applying noise to the weights and even the outputs of

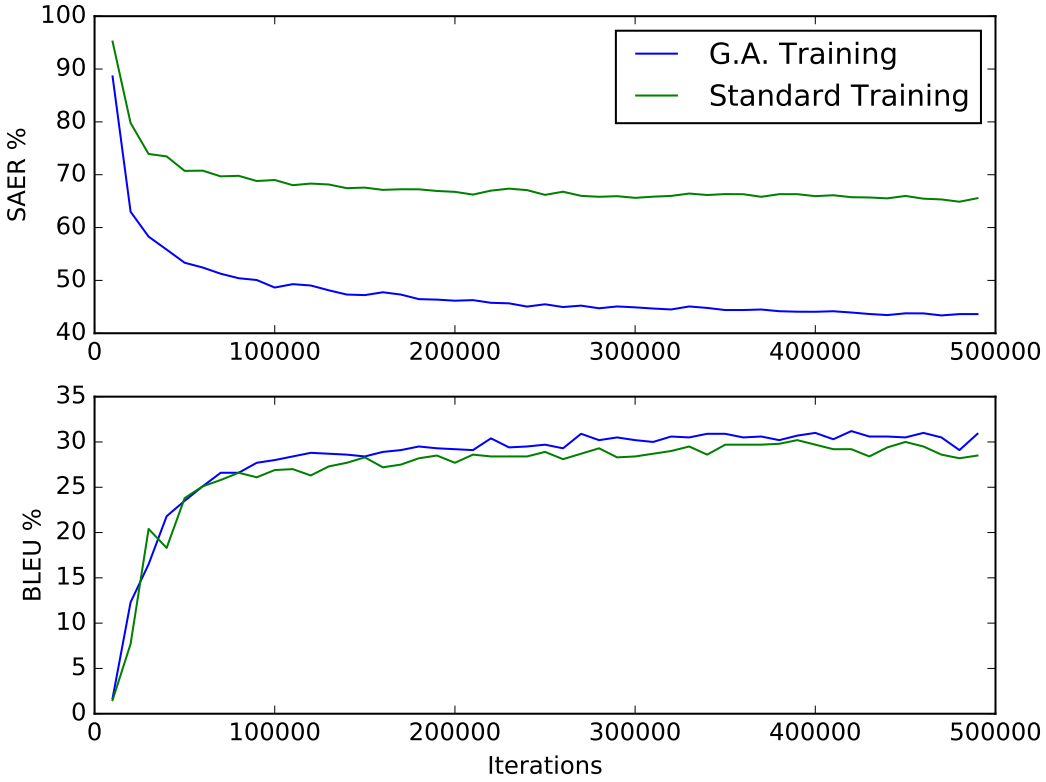

Figure 6.2: Comparison of guided alignment and standard training in terms of SAER and BLEU-score.

the corresponding network components. Unfortunately, none of these techniques showed positive results.

Our approach to prevent this problem was to give the attention an incentive to create an alignment by applying guided alignment training. We found two configurations that led to an excellent alignment. Fixed cost weights $\lambda_{\text{align}} = 5$ and $\lambda_{\text{CE}} = 1$ were supposed to reinforce the importance of alignment throughout the whole training. This configuration already lead to an alignment that was 2.6% better in AER than GIZA++. This result could still be beaten when we use cost weights that change over time during training. We achieved the best AER when we started with $\lambda_{\text{align}} = 1$ and $\lambda_{\text{CE}} = 0.001$ and uniformly added 0.001 to λ_{CE}

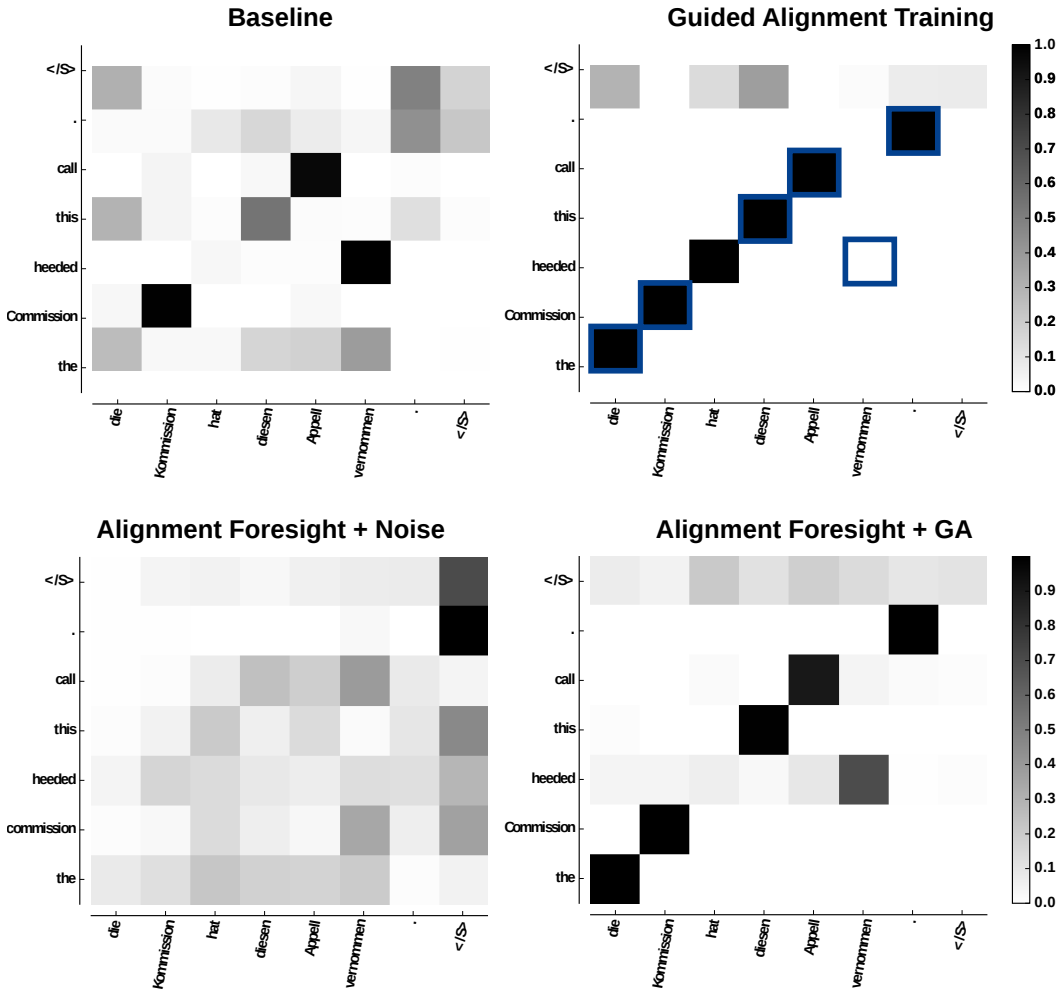

Figure 6.3: Attention weight matrices visualized in heatmap form.
Top-Left: Baseline attention-based NMT alignment.
Top-Right: Baseline + guided alignment training (GA).
GIZA-alignment points highlighted.
Bottom-Left: Baseline + AF + noise on attention output.
Bottom-Right: Baseline + AF + GA + dynamic changing of λ_{CE}.

during training to reach $\lambda_{\text{CE}} = 1$ in the last iteration. That means we start by training the attention alone and first reach a significant influence of $\mathcal{L}_{\text{CE}}$ when the network already should have learned a strong attention mechanism. With this configuration, we achieved an AER of 19.00% which is 3.78% lower than alignments created by GIZA++. However, at the same time, the SAER score is 6% higher than that we obtain with GIZA-alignments. This is most likely due to the fact that we evaluate hard alignments, as they are produced by the human annotator, against soft alignments in this instance. Since soft alignments sometimes tend to be evenly distributed over multiple alignment points, the direct comparison against a hard alignment is not a fair evaluation. Therefore we generate the corresponding hard alignments from the attention weights and obtain an SAER that is 10% better than its soft equivalent. Now, with a fairer comparison, the generated alignment beats the GIZA-alignment for all evaluation methods. Unfortunately, these results have to be regarded with caution, since we tuned the cost weights on the same data we use for evaluation. This is not avoidable since we only have a very limited number of manually aligned sentences. Nevertheless, we obtain an alignment that is clearly superior to the baseline alignments and also to the standard guided alignment approach as can be seen in figure 6.3. Also noteworthy in this experiment is the fact that both hard decisions and the symmetrization through merging help the alignment quality. This will be further discussed in the outlook (section 7.2) of this thesis.

The original motivation for the alignment-foresight approach was to generate a better alignment for guided alignment training. To test this, we used our best NMT (alignment) model to produce a soft alignment for the whole Europarl training corpus. Then, this data was used for guided alignment training to compare it to the guided alignment training with GIZA++ as target alignment. The results show that, even though the BLEU-score got slightly worse, we decreased the AER by 1.3%.

All these results show that attention-based models are in principle capable of generating excellent alignments, even though we still rely on GIZA++ to prevent the network from encoding target words in the attention weights for alignment foresight.

6.4 Translation Evaluation

The experiments we described in the previous chapter were conducted with the goal in mind that we want to generate a better alignment with our network. The following experiments will focus on the translation quality as it is the ultimate goal in MT to optimize this. So we run experiments for all the advanced attention

Europarl De-En	alignment-test				
Model	BLEU%	TER%	AER%	F1 % ($\varphi = 0.5$)	SAER %
GIZA++	-	-	22.7	74.9	28.2
Attention-Based	28.2	57.7	38.1	60.5	63.6
+ AF + GA $\lambda_{\text{align}} = 5$, $\lambda_{\text{CE}} = 1$	*82.3*	*8.6*	20.0	77.4	32.6
+ AF + GA $\lambda_{\text{align}} = 1$, $\lambda_{\text{CE}} = 0.001, \ldots, 1.0$	*87.2*	*5.9*	19.0	78.4	34.9
+ hard merged $j \rightarrow i$, $j \leftarrow i$	-	-	19.0	78.4	24.6
+ hard $j \rightarrow i$	-	-	20.6	77.0	25.9
+ hard $j \leftarrow i$	-	-	23.6	74.0	29.0
+ GA (GIZA++)	28.7	57.3	29.8	68.5	38.0
+ GA (AF-alignment)	28.3	57.5	28.5	69.8	36.7

Table 6.1: Alignment evaluation for alignment foresight models and comparison with GIZA-alignments.
Note: The BLEU-scores for alignment foresight are achieved with knowledge of the target word.

methods we described earlier on the IWSLT2013 and WMT2016 tasks to determine their value for NMT.

6.4.1 Alignment Feedback

To evaluate the convolutional alignment feedback approach, we searched for an optimal setting for the window size $2 \cdot k+1$ and the number of filters M we wanted to apply. Since the experiments on the full IWSLT data set are very costly and usually take more than one week, we did some prior experiments on the indomain set alone to find the best configurations for these hyperparameters. Some configurations turned out to be suboptimal, e.g. taking many filters ($M = 5$ and $M = 5$) with a large filter size ($k = 5$ and $k = 10$), even though none of the experiments lead to results below the indomain baseline. From these preliminary results, we selected the most promising ones. This led to experiments with a small filter size in combination with many filters ($k = 2$, $M = 5$) and experiments with a single large filter for convolution ($k = 10$, $M = 1$).

Both configurations led to great improvements of more than one BLEU on all test sets of the WMT task (table 6.3). The approach with multiple small filters ($k = 2$, $M = 5$) was even slightly better than applying one large window ($k = 10$, $M = 1$) and resulted in an average improvement of 1.9 BLEU on the WMT test sets. Both

IWSLT De-En	dev		test		eval11		alignment-test	
Model	BLEU%	TER%	BLEU%	TER%	BLEU%	TER%	AER%	SAER%
Attention-Based	30.5	48.7	29.3	50.6	33.9	46.6	41.8	66.3
+ conv ($k = 2$,$M = 5$)	30.8	49.0	29.5	50.2	34.3	45.8	41.3	66.6
+ conv ($k = 10$,$M = 1$)	31.4	49.6	29.5	50.7	33.2	47.1	41.8	67.6
+ bid-feedback	29.6	48.5	28.2	49.8	33.1	44.5	41.9	65.6

Table 6.2: Alignment feedback experiments on IWSLT2013 De-En

WMT En-Ro	newsdev2016/1		newsdev2016/2		newstest2016	
Model	BLEU%	TER%	BLEU%	TER%	BLEU%	TER%
Attention-Based	19.8	62.0	21.3	58.1	20.3	60.4
+ conv ($k = 2$,$M = 5$)	21.4	61.5	23.8	56.6	22.0	60.2
+ conv ($k = 10$,$M = 1$)	21.0	61.4	23.5	56.9	21.4	60.1
+ bid-feedback	19.2	64.4	21.7	60	19.9	63.5

Table 6.3: Alignment feedback experiments on WMT2016 En-Ro

approaches did not show similar improvements on the IWSLT task (table 6.2). The small filter approach improved on all test sets by an average of 0.3 BLEU. The large single filter approach was on average only 0.1 BLEU better than the baseline while having its best result on the `dev` set, where it improved by 0.9 BLEU and its worst result on the `eval11` set where the BLEU score decreased by 0.7 BLEU compared to the baseline. One possible reason for the discrepancy between the two result on IWSLT and WMT might be the fact that the IWSLT data is already reordered. For data that is not reordered like WMT, the positional information, as it gets extracted by the convolutional alignment feedback, can be very helpful.

Computing the feedback component with a bidirectional LSTM, however, did not result in any improvement over the baseline. In fact, the feedback calculated in this way seemed to disturb the model, since the results are below the baseline on most test sets. It is possible that the scalar-valued attention weights did not contain enough information to apply an LSTM on it. Therefore, the information we expected to be useful for the network turned out to add noise to the alignment computation.

6.4.2 Recurrent Attention

The recurrent attention was implemented as an LSTM over the decoder time for our experiments. On the large IWSLT2013 task, we did not notice any improvement

IWSLT De-En	dev		test		eval11		alignment-test	
Model	BLEU%	TER%	BLEU%	TER%	BLEU%	TER%	AER%	SAER%
Attention-Based	30.5	48.7	29.3	50.6	33.9	46.6	41.8	66.3
+ Rec.-Attention	30.8	48.0	29.2	49.9	33.3	45.6	33.5	68.1
+ bid-feedback	30.6	49.2	29.7	49.9	33.1	47.3	33.4	67.2
+ MD-Attention	27.5	51.9	26.0	52.3	29.6	48.2	36.7	70.3

Table 6.4: Recurrent attention experiments on IWSLT2013 De-En

WMT En-Ro	newsdev2016/1		newsdev2016/2		newstest2016	
Model	BLEU%	TER%	BLEU%	TER%	BLEU%	TER%
Attention-Based	19.8	62.0	21.3	58.1	20.3	60.4
+ Rec.-Attention	20.7	60.8	22.0	57.2	21.0	60.2
+ conv ($k = 2$,$M = 5$)	21.1	61.4	23.7	56.6	21.7	60.0
+ conv ($k = 10$,$M = 1$)	21.2	61.1	23.3	56.6	21.7	60.3
+ bid-feedback	20.5	61.5	22.8	57.1	21.3	60.1

Table 6.5: Recurrent attention experiments on WMT2016 En-Ro

over the attention-based baseline. The same model led to an average improvement of 0.8 BLEU over all evaluation sets.

However, the multidimensional attention approach was not able to reproduce the baseline results, let alone improve them. The MDGRU we used for our implementation and the new dependencies it introduces seem to be too complex for the network to learn. This results in a much slower convergence to a lower BLEU score, as it can be seen in figure 6.4.

Looking at the alignment scores for both the LSTM and MDGRU attention approach, we once again see the phenomenon we discussed in section 6.3.3 that there is a discrepancy between AER and SAER. The AER got much better compared to the baseline while the SAER has worsened at the same time. As indicated by the translation evaluation and our prior experiments, the AER value is not trustworthy here. Therefore we can confidently assume that the LSTM and MDGRU attention as we applied it here did not help but rather hurt the alignment quality on the IWSLT task.

6.4.3 Guided Alignment Training

In section 6.3, we evaluated the influence of guided alignment training on alignment and translation quality during training and we used guided alignment training to gain large improvements for alignment quality on the Europarl `align-test` set. In the following, we will show the final results of guided alignment training on the

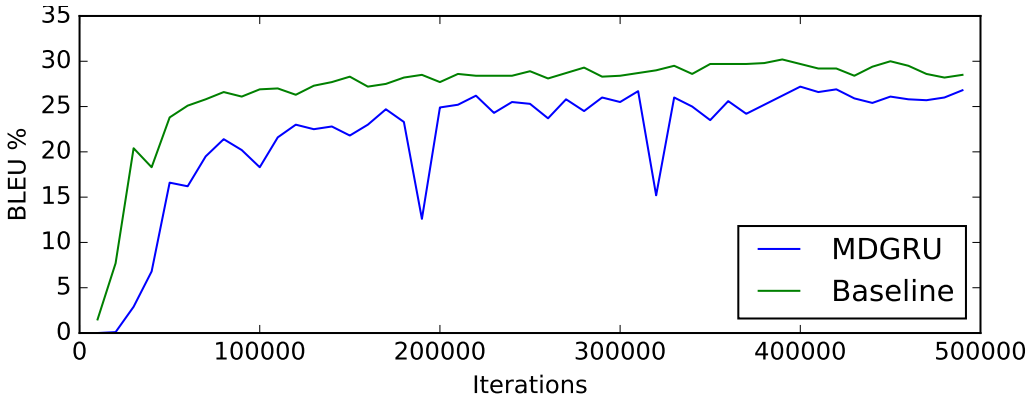

Figure 6.4: Comparison of translation quality during training for MDGRU attention and baseline model.

IWSLT De-En	dev		test		eval11		alignment-test	
Model	BLEU%	TER%	BLEU%	TER%	BLEU%	TER%	AER%	SAER%
Attention-Based	30.5	48.7	29.3	50.6	33.9	46.6	41.8	66.3
+ GA (GIZA++)	31.5	47.2	30.3	49.0	34.3	44.3	35.4	44.2
+ GA (AF)	31.4	46.9	29.8	49.5	34.7	44.4	35.0	43.1
+ AF + GA (GIZA++) $\lambda_{\text{align}} = 1$, $\lambda_{\text{CE}} = 0.01, \ldots, 1.0$	*81.3*	*9.5*	*82.1*	*8.4*	*83.3*	*7.5*	23.7	41.3

Table 6.6: Guided alignment experiments on IWSLT2013 De-En

IWSLT2013 and WMT2016 translation tasks as well as the alignment quality of the model resulting from training on full IWSLT data.

The results show a consistent improvement over all test sets for both tasks with an average increase of 0.8 BLEU on the IWSLT task and 1.7 BLEU on the WMT task. Also very noticeable is the fact that the improved alignment quality we noticed on the Europarl data (section 6.3.3) carries over on the IWSLT task. Here we see an improvement of 6% in AER as well as a drop of 22% in SAER.

We achieved a similar result when we replaced GIZA++ with the alignment foresight network, that we trained on IWSLT2013 data, to generate the target alignments for guided alignment training. The BLEU scores are almost the same as for guided alignment training with alignments produced by GIZA++, but we can see a slight improvement in AER (0.4%) and SAER (1.1%). These results clearly show that the soft alignment produced by our alignment foresight approach has a high quality that can also be used in practice to replace GIZA++.

WMT En-Ro	newsdev2016/1		newsdev2016/2		newstest2016	
Model	BLEU%	TER%	BLEU%	TER%	BLEU%	TER%
Attention-Based	19.8	62.0	21.3	58.1	20.3	60.4
+GA (GIZA++)	21.0	61.1	23.6	56.4	21.8	59.4

Table 6.7: Guided alignment experiments on WMT2016 De-En

All these results clearly indicate that guided alignment training introduces new valuable information to the network. Moreover, the network can use this information to improve the attention mechanism and thereby the alignment quality.

Chapter 7

Conclusions and Outlook

In this work, we investigated the attention-based neural machine translation approach with a specific focus on the attention mechanism, whose output can be interpreted as a soft alignment from source to target sentence. We developed different advanced attention methods that introduced additional information and dependencies into the network. These approaches were implemented and tested on two different data sets to evaluate the translation quality. Additionally, we evaluated and compared the alignment quality of different attention-methods as well as traditional statistical alignments generated by GIZA++.

7.1 Conclusions

The general goal of this thesis was to analyze, better understand and improve the attention-based soft alignments and thereby the overall translation quality. To enable this objective, we first had to analyze the attention mechanism, its alignment, and relation to the resulting translation. A first prior experiment that artificially generated attention-based alignments of different quality showed a high correlation between every alignment quality measure we evaluated and the translation quality measured in BLEU. Further experiments showed that we have a discrepancy between the SAER and AER as well as F-Measure in some cases for soft alignments. According to our assessments after looking at the generated alignments and as indicated by the corresponding BLEU scores, the SAER seems to be more reliable in practice when it comes to attention-based soft alignments.

Nevertheless, our experiments and the results we discussed in the related work section lead us to the conclusion that the alignment, generated by the attention mechanism, has a significant impact on the quality of the resulting translation.

Furthermore, we showed that attention-based alignments have room for improvement and that their alignment quality is poor in comparison to the current state-of-the-art alignments generated by GIZA++. However, we also showed that the attention-based NMT model is capable of generating alignments that outperform GIZA-alignments on our alignment test set.

The most successful approach that we tried was certainly the guided alignment training. For this, the results clearly showed that the GIZA-alignments introduce helpful information to the network when used as target alignment and thereby improve the translation and alignment quality very significantly. Another successful approach was the convolutional alignment feedback that we adapted from automatic speech recognition. This method leads to substantial improvements in translation quality, in particular on the WMT2016 data set. The second alignment feedback method that we proposed in this thesis, which implements a complete context representation by a bidirectional RNN, did not result in any noticeable improvements over the baseline NMT model.

Our experiments with a recurrent and a multidimensional attention mechanism showed once again that introducing additional dependencies that the network should benefit from in theory may not have the same expected effect in practice. The recurrent attention mechanism did not show any significant improvements on the IWSLT2013 task. However, the same model lead to an increased translation quality on the smaller WMT2016 task. This, the unsuccessful experiments with MD-attention on the IWSLT2013 task and the fact that we noticed improvements with both approaches during prior experiments on the small IWSLT2013 indomain task, lead us to the conclusion that the attention mechanism may still benefit from the proposed dependencies afterall. The problem with both approaches on large data sets is just that we do not know how we can train these approaches successfully.

7.2 Outlook

As we have mentioned in the previous section, our experiments showed that attention-based alignments still have room for improvements. We can beat the alignment scores with our models, but we still need GIZA as guidance to achieve this. That means that we are limited in some way by the quality of the GIZA-alignment. Therefore, one future goal would be to find some way to run alignment foresight models without GIZA++ while, at the same time, preventing the encoding of target words in the attention weights. Such a method needs to enforce some kind of hard alignment without destroying the information or the differentiability.

We also learned from our experiments that it makes a difference if we symmetrize alignments and convert soft alignments to hard alignments before computing the SAER. Therefore it would be interesting to see how these two operations influence the translation quality. Symmetrization already showed promising results [Cohn & Hoang$^+$ 16] when applied for the whole training. Semi-hard decisions, i.e. encouraging the network to maximize one or two attention points, would also be possible by applying noise on the attention outputs. Therefore this topic is certainly an interesting research direction for the future.

Since the convolutional alignment feedback leads to some gain in translation quality, it would be the next logical step to extend this one-dimensional convolution to a two-dimensional case. This would mean that we include the attention-weights from multiple time steps over decoder time in the convolutional operation.

As we have mentioned in our conclusions, the open question for recurrent and multidimensional attention is how we can train them. One idea for future work would be to utilize techniques like e.g. pretraining or guided alignment training to aid the training process of these complex models. And if the MD-attention approach proves to be successful, as we believe it will, we will need to increase the efficiency of the MDGRU or MDLSTM implementations, which may be possible by replacing it with a native CUDA implementation.

Appendix A

Appendix

A.1 Notation

A.1.1 Statistical Machine Translation

$Pr(\cdot\|\cdot)$	general distribution
$p(\cdot\|\cdot)$	concrete model of a distribution
$p_\theta(\cdot\|\cdot)$	concrete model of a distirbution with free paramter θ
e	english or target word
f	french or source word
a	alignment mapping from source to target
I	target sentence length
J	source sentence length
i	target sentence index
j	source sentence index
e_1^I	target sentence
f_1^J	source sentence
a_1^J	alignment covering all source positions
$\hat{e}_1^{\hat{I}}$	target hypothesis - translated sentence
$c(x)$	count of x
B	alignment mapping from target to source
B_i	set of source indices aligned to target position i
ϕ	fertility
$\tilde{f}_1^K$	source phrase segmentation
$\tilde{e}_1^K$	target phrase segmentation
K	number of phrases
λ	log-linear model parameter
h	model function

A.1.2 Artificial Neural Networks

x	arbitrary input
z	intermediate representation before applying an activation function
y	arbitrary output
w	arbitrary weight parameter
b	arbitrary bias parameter
σ	arbitrary non-linear activation function
W, U, V	arbitrary weight matrix
g	function parameterised by a neural network
$\overrightarrow{g}$	function parameterised by an RNN in forward direction
$\overleftarrow{g}$	function parameterised by an RNN in backwardd direction
k	window size for a convolutional or pooling operation
$*$	convolutional operation
c	LSTM cell-state
$\mathcal{X}$	set of data
$\mathcal{L}$	loss function
N	number of items in training set
δ	Knoecker delta
Δw	negateve error gradient w.r.t. w
μ	momentum parameter
η	learning rate
$E[\cdot]$	average
Δ	arbitrary error gradient

A.1.3 Neural Machine Translation

$\mathcal{V}$	vocabulary (set of words ω_v)
h	arbitrary hidden layer output (usually encoder related)
s	arbitrary hidden layer output (usually decoder related)
c	context vector representing source side information
g_{enc}	encoder function
g_{dec}	decoder function
g_{out}	ouptu function of the decoder
$\tilde{\alpha}$	soft alignment energy value between source and target
α	normalized soft alignment
β	annotation vector from past alignments
g_{rec}	arbitrary RNN function usually a GRU or LSTM
d	decay gate
sd	decay state
ψ	positional bias
ξ	arbitrary bias for attention computation
γ	alignment feedback vector
G	weight matrix for a convolutional filter
g_{md}	multidimensional RNN function
A	hard alignment (usually computed by GIZA++)
$\tilde{e}$	target word encoded by a projection layer
φ	trade-off between precision and recall
P	set of abiguous alignment points
S	set of unabiguous alignment points
M	matrix representation of an alignment

A.2 Corpus Statistics

A.2.1 WMT2016 En-Ro

		English	Romanian
train	Sentences	604K	
	Running Words	15481016	15786717
	Vocabulary	92326	128429
	OOV Rate with 30k short list	0.7%	1.8%
europarl_test	Sentences	2000	
	Running Words	52077	52575
	Vocabulary	4837	6890
	OOVs (Rate)	103 (0.2%)	203 (0.4%)
	OOVs with 30k short list (Rate)	344 (0.7%)	845 (1.6%)
newsdev2016_1	Sentences	1000	
	Running Words	24731	26713
	Vocabulary	5082	6408
	OOVs (Rate)	938 (3.8%)	1504 (5.6%)
	OOVs with 30k short list (Rate)	1602 (6.5%)	2987 (11.2%)
newsdev2016_2	Sentences	999	
	Running Words	25228	25665
	Vocabulary	4708	6480
	OOVs (Rate)	733 (2.9%)	1296 (5.0%)
	OOVs with 30k short list (Rate)	1289 (5.1%)	2992 (11.7%)

A.2.2 IWSLT2013 De-En

		German	English
train (full data)	Sentences	4.3M	
	Running Words	108M	108M
	Vocabulary	836K	792K
train (in-domain)	Sentences	138K	
	Running Words	2.6M	2.7M
	Vocabulary	75K	50K
dev	Sentences	887	
	Running Words	20K	20.1K
	Vocabulary	4.1K	3.3K
	OOVs with full vocabulary	468 (2.3%)	197 (0.9%)
	OOVs with 30K shortlist	1346 (6.7%)	656 (3.3%)
eval	Sentences	1436	
	Running Words	27.2K	27.6K
	Vocabulary	4.6K	3.7K
	OOVs with full vocabulary	449(1.6%)	1110(4.1%)
	OOVs with 30K shortlist	1526 (5.6%)	1716 (6.5%)
test	Sentences	1565	
	Running Words	31.6K	32.6K
	Vocabulary	5.0K	3.9K
	OOVs with full vocabulary	677 (2.1%)	1377 (4.4%)
	OOVs with 30K shortlist	1811 (5.7%)	2000 (6.4%)

A.2.3 Europarl De-En

		German	English
train (full data)	Sentences	1.2M	
	Running Words	32M	34M
	Vocabulary	305K	100K
align-test	Sentences	504	
	Running Words	9.9K	10.3K
	Vocabulary	2.8K	2.4K
	OOVs with full vocabulary	6 (0.1%)	1 (0.0%)
	OOVs with 30K shortlist	276 (2.8%)	50 (0.5%)

List of Figures

List of Tables

Bibliography

[Ayan & Dorr 06] N. F. Ayan, B. J. Dorr. Going beyond aer: An extensive analysis of word alignments and their impact on mt. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pp. 9–16. Association for Computational Linguistics, 2006.

[Bahdanau & Cho$^+$ 15] D. Bahdanau, K. Cho, Y. Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, May 2015.

[Bengio & Ducharme$^+$ 03] Y. Bengio, R. Ducharme, P. Vincent, C. Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, Vol. 3, pp. 1137–1155, February 2003.

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

[Bengio & Simard$^+$ 94] Y. Bengio, P. Simard, P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, Vol. 5, No. 2, pp. 157–166, 1994.

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

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

[Brown & Pietra$^+$ 93] P. F. Brown, V. J. D. Pietra, S. A. D. Pietra, R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Comput. Linguist.*, Vol. 19, No. 2, pp. 263–311, June 1993.

[Cettolo & Girardi$^+$ 12] M. Cettolo, C. Girardi, M. Federico. Wit^3: Web inventory of transcribed and translated talks. In *Proceedings of the 16th Conference of the European Association for Machine Translation (EAMT)*, pp. 261–268, Trento, Italy, May 2012.

[Chen & Matusov+ 16] W. Chen, E. Matusov, S. Khadivi, J.-T. Peter. Guided alignment training for topic-aware neural machine translation. *arXiv preprint arXiv:1607.01628*, 2016.

[Cho & van Merrienboer+ 14] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734. Association for Computational Linguistics, October 2014.

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

[Chung & Gulcehre+ 14] J. Chung, C. Gulcehre, K. Cho, Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, December 2014.

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

[Crammer & Singer 03] K. Crammer, Y. Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, Vol. 3, No. Jan, pp. 951–991, 2003.

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

[dos Santos & Zadrozny 14] C. N. dos Santos, B. Zadrozny. Learning character-level representations for part-of-speech tagging. In *ICML*, pp. 1818–1826, 2014.

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

[Elman 90] J. L. Elman. Finding structure in time. *Cognitive Science*, Vol. 14, No. 2, pp. 179 – 211, 1990.

[Feng & Liu+ 16] S. Feng, S. Liu, M. Li, M. Zhou. Implicit distortion and fertility models for attention-based encoder-decoder nmt model. *arXiv preprint arXiv:1601.03317*, 2016.

[Fraser & Marcu 07] A. Fraser, D. Marcu. Measuring word alignment quality for statistical machine translation. *Computational Linguistics*, Vol. 33, No. 3, pp. 293–303, 2007.

[Fukushima 80] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, Vol. 36, No. 4, pp. 193–202, 1980.

[Goodman 01] J. Goodman. Classes for fast maximum entropy training. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, Vol. 1, pp. 561–564. IEEE, 2001.

[Graves & Fernández$^+$ 07] A. Graves, S. Fernández, J. Schmidhuber. *Multi-dimensional Recurrent Neural Networks*, pp. 549–558. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[Graves & Schmidhuber 05] A. Graves, J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, Vol. 18, No. 5, pp. 602–610, August 2005.

[Graves 12] A. Graves. Supervised sequence labelling with recurrent neural networks. In *Supervised Sequence Labelling with Recurrent Neural Networks*, pp. 15–35. Springer, 2012.

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

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

[Hochreiter & Bengio$^+$ 01] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependenciess. *A field guide to dynamical recurrent neural networks. IEEE Press*, 2001.

[Hochreiter & Schmidhuber 97a] S. Hochreiter, J. Schmidhuber. Long short-term memory. *Neural computation*, Vol. 9, No. 8, pp. 1735–1780, November 1997.

[Hochreiter & Schmidhuber 97b] S. Hochreiter, J. Schmidhuber. Lstm can solve hard long time lag problems. *Advances in neural information processing systems*, pp. 473–479, 1997.

[Hochreiter 91] S. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, pp. 91, 1991.

[Hu & Lu+ 14] B. Hu, Z. Lu, H. Li, Q. Chen. Convolutional neural network architectures for matching natural language sentences. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pp. 2042–2050. Curran Associates, Inc., 2014.

[Hubel & Wiesel 68] D. H. Hubel, T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, Vol. 195, No. 1, pp. 215–243, 1968.

[Jordan 86] M. I. Jordan. Attractor dynamics and parallellism in a connectionist sequential machine, 1986.

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

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

[Krizhevsky & Sutskever+ 12] A. Krizhevsky, I. Sutskever, G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[Lang & Waibel+ 90] K. J. Lang, A. H. Waibel, G. E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, Vol. 3, No. 1, pp. 23 – 43, 1990.

[LeCun & Bengio+ 15] Y. LeCun, Y. Bengio, G. Hinton. Deep learning. *Nature*, Vol. 521, No. 7553, pp. 436–444, May 2015.

[LeCun & Boser+ 89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, Vol. 1, No. 4, pp. 541–551, 1989.

[LeCun & Bottou+ 12] Y. A. LeCun, L. Bottou, G. B. Orr, K.-R. Müller. *Efficient BackProp*, pp. 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[Leifert & Strauß+ 14] G. Leifert, T. Strauß, T. Grüning, R. Labahn. Cells in multidimensional recurrent neural networks. *arXiv preprint arXiv:1412.2620*, 2014.

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

[Mi & Sankaran+ 16] H. Mi, B. Sankaran, Z. Wang, A. Ittycheriah. A coverage embedding model for neural machine translation. *arXiv preprint arXiv:1605.03148*, 2016.

[Mi & Wang+ 16] H. Mi, Z. Wang, A. Ittycheriah. Supervised attentions for neural machine translation. *arXiv preprint arXiv:1608.00112*, 2016.

[Mohamed & Dahl+ 12] A.-r. Mohamed, G. E. Dahl, G. Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, Vol. 20, No. 1, pp. 14–22, 2012.

[Nair & Hinton 10] V. Nair, G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814, 2010.

[Ney & Essen+ 95] H. Ney, U. Essen, R. Kneser. On the estimation of 'small' probabilities by leaving-one-out. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 17, No. 12, pp. 1202–1212, Dec 1995.

[Ney 16] H. Ney. *Lecture Notes Statistical Classification.* 2016.

[Nickolls & Buck+ 08] J. Nickolls, I. Buck, M. Garland, K. Skadron. Scalable parallel programming with cuda. *Queue*, Vol. 6, No. 2, pp. 40–53, 2008.

[Och & Ney 03] F. J. Och, H. Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, Vol. 29, No. 1, pp. 19–51, 2003.

[Och & Ney 04] F. J. Och, H. Ney. The alignment template approach to statistical machine translation. *Computational linguistics*, Vol. 30, No. 4, pp. 417–449, 2004.

[Och 03] F. J. Och. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pp. 160–167, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[Papineni & Roukos+ 02] K. Papineni, S. Roukos, T. Ward, W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 311–318. Association for Computational Linguistics, July 2002.

[Ren & Kiros+ 15] M. Ren, R. Kiros, R. Zemel. Exploring models and data for image question answering. In *Advances in Neural Information Processing Systems*, pp. 2953–2961, 2015.

[Robinson & Fallside 87] A. Robinson, F. Fallside. *The utility driven dynamic error propagation network.* University of Cambridge Department of Engineering, 1987.

[Rumelhart & Hinton$^+$ 86] D. Rumelhart, G. Hinton, R. Williams. Learning internal representation by back propagation. *Parallel distributed processing: exploration in the microstructure of cognition*, Vol. 1, October 1986.

[Sankaran & Mi$^+$ 16] B. Sankaran, H. Mi, Y. Al-Onaizan, A. Ittycheriah. Temporal attention model for neural machine translation. *arXiv preprint arXiv:1608.02927*, 2016.

[Seide & Li$^+$ 11] F. Seide, G. Li, D. Yu. Conversational speech transcription using context-dependent deep neural networks. In *Interspeech*, pp. 437–440, 2011.

[Snover & Dorr$^+$ 06] M. Snover, B. Dorr, R. Schwartz, L. Micciulla, J. Makhoul. A study of translation edit rate with targeted human annotation. In *Proceedings of association for machine translation in the Americas*, Vol. 200, 2006.

[Srivastava & Hinton$^+$ 14] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, Vol. 15, No. 1, pp. 1929–1958, 2014.

[Sutskever & Vinyals$^+$ 14] I. Sutskever, O. Vinyals, Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 3104–3112, 2014.

[Szegedy & Liu$^+$ 15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

[Tu & Lu$^+$ 16] Z. Tu, Z. Lu, Y. Liu, X. Liu, H. Li. Modeling coverage for neural machine translation. In *54th Annual Meeting of the Association for Computational Linguistics*, August 2016.

[Van Merriënboer & Bahdanau$^+$ 15] B. Van Merriënboer, D. Bahdanau, V. Dumoulin, D. Serdyuk, D. Warde-Farley, J. Chorowski, Y. Bengio. Blocks and fuel: Frameworks for deep learning. *arXiv preprint arXiv:1506.00619*, 2015.

[Vilar & Popovic$^+$ 06] D. Vilar, M. Popovic, H. Ney. Aer: do we need to "improve" our alignments? In *IWSLT*, pp. 205–212, 2006.

[Vogel & Ney$^+$ 96] S. Vogel, H. Ney, C. Tillmann. Hmm-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pp. 836–841. Association for Computational Linguistics, 1996.

[Werbos 88] P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, Vol. 1, No. 4, pp. 339 – 356, May 1988.

[Werbos 90] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, Vol. 78, No. 10, pp. 1550–1560, October 1990.

[Weston & Chopra$^+$ 14] J. Weston, S. Chopra, A. Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[Williams & Zipser 95] R. J. Williams, D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures and applications*, pp. 433–486, 1995.

[Xu & Ba$^+$ 15] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, Vol. 2, No. 3, pp. 5, 2015.

[Yao & Cohn$^+$ 15] K. Yao, T. Cohn, K. Vylomova, K. Duh, C. Dyer. Depth-gated lstm. In *Presented at Jelinek Summer Workshop on August*, Vol. 14, 1, 2015.

[Zeiler 12] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[Zhang & Xiong$^+$ 16] B. Zhang, D. Xiong, J. Su. Recurrent neural machine translation. *arXiv preprint arXiv:1607.08725*, 2016.