Abstract

Currently, most approaches to neural machine translations use an encoder that produces continuous representations of the words in the input sentence. Next, another component decodes the representations into output words, where a separate attention mechanism typically allows the model to attend to different input words depending on the previous predictions. This approach comes with two potential problems. First, the input representations remain constant during the prediction process, which may not provide enough variance for the model to effectively discriminate between them. Second, a high burden is put on the component responsible for predicting output words (decoder), as translations involves many subtasks: knowing which words to translate next, remembering which words have been translated, producing a fluent and grammatically correct sentence, and more. As a result, many neural models suffer from over-translation (generating unnecessary words), under-translation (forgetting to translate words) and repetition.

By extending such models with a re-encoding component, which allows the model to update the input representations depending on the previous predictions, these tasks can naturally be moved away from the decoding component. This thesis investigates two architectures that use re-encoding and compares them to multiple baselines. The qualitative and quantitative results show that re-encoding can potentially improve performance of neural models, especially on longer sentences.
# Contents

Abstract iii

1 Introduction 1

2 Background 9
  2.1 Recurrent Neural Networks . . . . . . . . . . . . . . . . . . . . 9
  2.2 Long Short-Term Memory . . . . . . . . . . . . . . . . . . . . . 17
  2.3 Encoder-Decoder Networks . . . . . . . . . . . . . . . . . . . . 18
  2.4 Grid Long Short-Term Memory . . . . . . . . . . . . . . . . . . 22
  2.5 Active Memory & Re-encoding . . . . . . . . . . . . . . . . . . . 25

3 Models 27
  3.1 Model I: Grid Re-encoder . . . . . . . . . . . . . . . . . . . . . 27
    3.1.1 Re-encoder . . . . . . . . . . . . . . . . . . . . . . . . . . 28
    3.1.2 Decoder . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
  3.2 Model II: Grid Encoder-Decoder . . . . . . . . . . . . . . . . . . 35
    3.2.1 Bi-directional Encoder . . . . . . . . . . . . . . . . . . . . 36
    3.2.2 Decoder . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

4 Experiments 43
  4.1 Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
  4.2 Models . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
  4.3 Training . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
  4.4 Evaluation Metrics . . . . . . . . . . . . . . . . . . . . . . . . . . 46
  4.5 Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
    4.5.1 Finding the Optimal Model . . . . . . . . . . . . . . . . . 47
    4.5.2 Translation Performance . . . . . . . . . . . . . . . . . . . 50

5 Conclusion 61
Chapter 1

Introduction

The recent emergence of neural models in the field of natural language processing has been very influential. It has transformed the field, which is now almost completely dominated by neural models. Neural models have been shown to be effective at many tasks, including language modeling [Bengio et al., 2003, Mikolov et al., 2010], part-of-speech tagging, chunking, named entity recognition and semantic role labeling [Collobert and Weston, 2008, Collobert et al., 2011], as well as neural machine translation [Kalchbrenner and Blunsom, 2013, Sutskever et al., 2014, Cho et al., 2014]. They are powerful competitors to the traditional approaches and it is clear that they are here to stay.

In this thesis we identify a potential shortcoming of existing neural architectures for machine translation: neural models have a large memory at their disposal, but are limited in their flexibility to use and update this memory. This thesis proposes two novel additions to existing models to overcome this shortcoming. Before we dive into the details, we will first have a quick overview of the history of machine translation and evaluate the current situation.

A Short History of Machine Translation

The earliest approaches to machine translations were rule-based [Hutchins, 2007]. Such approaches were governed by lexical, syntactic, and morphological rules, among others. They would directly translate a sentence into the target language, or first translate it into an intermediate representation and then translate that to the target language. The advantage of rule-based systems was that they did not need any data. The rules were hand-written by linguistic experts and so could potentially produce perfect translations. However, writing the rules required extensive linguistic knowledge, and doing it by hand was time-consuming. In the end, it turned out that this approach was not scalable to real-world machine translation applications.

In the 1980s, data-driven approaches, that typically rely on parallel corpora, started to take over. A parallel corpus is a dataset with sentences in the source
Chapter 1. Introduction

Figure 1.1: A parallel corpus contains translations of sentences in at least two different languages.

language and translations in the target language (Figure 1.1). Example-based approaches [Nagao, 1984] looked up sentences similar to the input sentence in a parallel corpus and modified the reference translation to produce an output translation. However, modifying reference translations of the selected example sentences in order to produce fluent and grammatical output was a problematic challenge in the example-based approach.

Not long after that statistical models for machine translations were proposed. Initially, these models were word-based [Brown et al., 1988, Brown et al., 1990], meaning that the statistics were derived from individual word frequencies. The statistical approach views machine translation as a statistical optimization problem. Statistical models typically consist of two components: a translation model and a language model. Together they are used to optimize the conditional probability of the target sentence \( \mathbf{y} = y_1, ..., y_T \) given the source sentence \( \mathbf{x} = x_1, ..., x_S \):

\[
p(y|x) = \frac{p(x|y)p(y)}{p(x)} \propto p(x|y)p(y) \quad (1.1)
\]

To find the best translation \( \hat{y} \), we formulate the following optimization problem:

\[
\hat{y} = \arg\max_{y} p(x|y)p(y) \quad (1.2)
\]

\( p(y) \) corresponds to fluency in the target language and is computed by the language model. It estimates the probability of observing a sequence of words \( \mathbf{y} \). It is only used to model the target language and can in principle be trained on any corpus in the target language.

\( p(x|y) \) corresponds to faithfulness and is computed by the translation model. The translation model estimates the probability that any sequence of words \( \mathbf{x} \)
translates into sequence of target words $y$. The probabilities are roughly estimated by counting co-occurrences of words in the parallel corpus: if one source word frequently co-occurs with a specific target word, then it is likely they are translations of each other. Typically, the alignments between source and target words are treated as a latent variable.

The early models were based on word counts, without taking into account context. However, the word-based approach is too simplistic, because the translation of a word often depends on surrounding words, which word-based models cannot capture. To solve this problem, more sophisticated phrase-based models were proposed [Och et al., 1999, Zens et al., 2002, Koehn et al., 2003]. Phrase-based models improve over word-based models by estimating the statistics of phrase alignments (sub-sequences of words) instead of word alignments. By using phrases, the model can incorporate context into the translation probabilities. One of the challenges in this approach is sparsity: the longer a phrase, the less likely it is to appear in a corpus, and thus the less reliable the estimation of its translation probability. In hierarchical phrase-based models [Chiang, 2007] phrases can consist of smaller sub-phrases. The general idea is that phrases can contain placeholders which can be filled in by other phrases. The composition of phrases is learned by a separate model that produces rules consisting of a phrase in the source language and a matching phrase in the target language. This allows the phrases to be more general, so that the observed statistics are likely more reliable, reducing the problem of sparsity.

The main advantage of the statistical approach is that it is data-driven and thus requires little linguistic knowledge, contrary to the rule-based approach, which requires an enormous number of rules that are difficult to create and maintain. Statistical models can be trained automatically on large amounts of data and require little hand-tuning by human experts. However, statistical models also come with some important issues.

Statistical methods typically model fragments of natural language. Fragments are typically $n$-grams, which are consecutive sequences of words or characters of length $n$. In natural language most fragments become more rare as their length increases, which makes the estimation of their statistics less reliable. To address this, an upper limit can be placed on the size of the fragments, but this means that the model has to make strong assumptions about independence between fragments. Natural language is never independent, so this assumption is problematic.

Moreover, statistical methods typically consist of a cascade of components, where the output of one component is used as input to the next. Some of these components may be generative, by trying to model certain aspects of the data, such as syntax. Then, the final component which uses the other components and produces the translations may be trained discriminatively on a parallel corpus. This means that many of the components were not trained to do machine translation, but rather to do some other task. The result is a potentially suboptimal
cascade of components where individual components may not perform well in a machine translation task.

**Neural Machine Translation**

Recently, neural networks have started to gain popularity in the natural language processing community. Neural machine translation has become a competitive alternative to the traditional statistical approach [Kalchbrenner and Blunsom, 2013, Sutskever et al., 2014, Cho et al., 2014, Bahdanau et al., 2015]. In contrast to statistical machine translation, where many components are built and trained separately, in the neural approach a single model is built and trained discriminatively in an end-to-end fashion. Neural networks are also able to directly model the compositionality of natural language without making any assumptions about the independence of fragments.

The neural approach addresses some of the issues of the statistical approach, but it has not always been feasible. Its adoption by the general natural language community is still recent and was made possible by a number of important inventions and discoveries, including the following:

- The neural language model [Bengio et al., 2003], which improves over traditional n-gram-based and feature-based language models and showed that neural networks are applicable in natural language processing tasks.

- The neural attention mechanism [Bahdanau et al., 2015, Luong et al., 2015], which allows the model to selectively attend to parts of the source sentence and greatly improves translation quality of longer sentences.

- The Long Short Term Memory unit [Hochreiter and Schmidhuber, 1997, Gers et al., 2000] and related units such as Gated Recurrent Unit [Cho et al., 2014], which greatly improve learning long-distance temporal relationships.

- Better initialization strategies for deep neural networks such as the one proposed by [Glorot and Bengio, 2010] that make sure the weights of the neural network start in the right range depending on the type of activation functions and sizes of the network’s layers.

- More advanced optimization methods that improve learning efficiency, a popular choice being Adam [Kingma and Ba, 2014].

- The development of new hardware that makes massive parallel computation more feasible and the support for GPUs in programming frameworks.

Most early neural architectures for machine translation use recurrent neural networks. Specifically, the **encoder-decoder** framework is a common architecture (typically) consisting of two recurrent neural networks, an encoder and a decoder.
The encoder processes the source sentence sequentially and outputs fixed-size representations of the source sentence called context vectors. The decoder uses these context vectors to predict outputs. The decoder is autoregressive, which means that in subsequent steps, the decoder receives as input its previous output. The decoding process is repeated until the output sentence is complete.

In the neural approach, the conditional probability of a target sentence $y = y_1, \ldots, y_T$ given the source sentence $x = x_1, \ldots, x_S$ is modeled as follows:

$$p(y|x) = \prod_{t=1}^{T} p(y_t|x, y_1, \ldots, y_{t-1})$$

In other words, the probability of each target word is dependent on the entire source sentence and the target words predicted so far.

**Contributions of this Thesis**

Most neural models, like the encoder-decoder, produce source sentence representations (encodings) that remain constant while generating the target sentence (decoding). In this approach, the encodings are computed independently of the target sentence. Some flexibility can be added by using an attention mechanism [Bahdanau et al., 2015], which allows the model to selectively attend to the source encodings at each decoding step. Roughly, an attention mechanism computes a weighted sum of the encodings at each decoding step and uses it as input to the decoder. This mechanism is motivated by the fact that depending on where the model is in generating the target sentence, different parts of the source sentence may be more relevant. An attention mechanism allows the model to focus only on the relevant parts of the source sentence and ignore less relevant parts, which can greatly improve translation quality, especially when sentences get longer.

Although neural attention can be very effective when it comes to translating longer sentences, it is not perfect and its standard formulation might be too simplistic. It is possible that the weighted encodings produced by the attention mechanism do not give the model enough flexibility to properly discriminate between source words due to a lack of variance [Zhang et al., 2017]. Another potential problem is that the standard attention mechanism still puts too much of a burden on the decoder. This sometimes leads to over-translation (generating unnecessary words), under-translation (forgetting to translate words) and repetition. Tasks such as modeling coverage (i.e., remembering which parts of the source sentence have already been translated) and knowing which words to translate next may be more naturally carried out by a separate mechanism [Yang et al., 2016, Tu et al., 2016, Cohn et al., 2016].

The aforementioned problems could be solved by introducing a re-encoding component, with which the model gains full flexibility to update the encodings
before each prediction and to incorporate information about the target sentence generated so far. In broad terms, a re-encoding component may be a neural network that takes as input an encoding and the state of the decoder, and produces as output a re-encoded source sentence representation.

This thesis builds on the work of [Kalchbrenner et al., 2015], which proposes a special type of neural unit, the Grid Long Short-Term Memory (Grid LSTM), and a neural architecture for machine translation that uses re-encoding. The Grid LSTM is a generalization of the standard LSTM [Hochreiter and Schmidhuber, 1997] that allows the unit to be laid out in a multi-dimensional grid, and can be used in a multitude of useful network architectures. The main contribution of this thesis is the investigation and proposal of two models for machine translation that combine Grid LSTM, re-encoding, and neural attention.

The first model, the Grid Re-encoder, based on Kalchbrenner’s Re-encoder model, views translation as a two-dimensional mapping from source sentence to target sentence. This deep neural model revisits the entire source sentence for each target prediction and can thus implicitly attend to relevant parts depending on the previous predictions. While this model is able to implicitly attend to parts of the source sentence by re-encoding, it could be more natural to give it the option to also do so explicitly. This is why the Grid Re-encoder is extended with an additional attention mechanism that enables the model to explicitly attend to parts of its memory after revisiting (re-encoding) the source sentence, depending on the previous predictions.

The second model, the Grid Encoder-Decoder, is an encoder-decoder network with Grid LSTM units and an active attention mechanism. The attention mechanism is explicit by definition. However, as opposed to the Grid Re-encoder, a standard attention mechanism only allows the model to manipulate its memory to a limited extent by computing a weighted sum of the encodings. For the Grid Encoder-Decoder, the novelty lies in the active attention mechanism: rather than simply re-using encodings that remain constant, the active attention mechanism allows the model to update (re-encode) the encodings before using them at each decoding step.

Both models are evaluated on two corpora. They are tested in various configurations and their performance is investigated quantitatively and qualitatively. Based on the results, we conclude that re-encoding can indeed be beneficial, especially on longer sentences.

Structure of this Thesis

We will first introduce the background of this thesis and explain the basics of recurrent neural networks and neural architectures in Chapter 2. In this chapter, we look at how recurrent neural networks work, how they are trained, and how they are used in neural architectures for machine translation. It is necessary to have a strong understanding of these underlying concepts in order to understand the
inner workings of the models that are proposed and explained in detail in Chapter 3. In Chapter 4 we evaluate the effectiveness of the models and compare them to multiple baselines, and find that re-encoding improves upon traditional models without re-encoding. Finally, the thesis is wrapped up with final conclusions in Chapter 5.

Notation

Throughout this thesis, the following conventions for mathematical notation are used:

- Symbols and scalars are denoted as plain letters: $x$.
- Vectors are denoted using bold face: $\mathbf{x}$.
- Matrices are denoted using uppercase bold face: $\mathbf{W}$.
- Element-wise multiplication of vectors is denoted by $\odot$.
- Vector concatenation is denoted by $[x; y]$.
- Time indices are generally denoted using a $t$ subscript: $x_t$
- When a model deals with time indices for both a source and target sentence, the $s$ subscript is used to denote time steps on the source side and the $t$ subscript is used to denote time steps on the target side.
- Superscripts are typically used for disambiguation purposes: $\mathbf{W}^{\text{encoder}}$ vs. $\mathbf{W}^{\text{decoder}}$. 
Chapter 2

Background

In order to fully understand the details of the models proposed in Chapter 3, it is necessary to have a strong understanding of the basic concepts recurrent neural networks and how they can be used in neural machine translation. This chapter provides the necessary context. We will cover the basics of recurrent neural networks and see how they are applied in sequence-to-sequence modeling tasks such as machine translation. We will also go over specific, but popular neural architectures that the models proposed in this thesis are based on. Finally, we explore how we can modify existing architectures to allow more flexible use of the memory.

2.1 Recurrent Neural Networks

Recurrent neural networks [Elman, 1990] are used for modeling sequences. They process inputs sequentially and produce an output for each input. Their internal state, or memory, allows them to capture temporal relationships between parts of the input sequences, which makes them especially suitable for natural language processing tasks which involve sequences, such as machine translation [Kalchbrenner and Blunsom, 2013, Sutskever et al., 2014, Cho et al., 2014] and language modeling [Bengio et al., 2003, Mikolov et al., 2010].

In recurrent neural networks, inputs \( x_t \) of an input sequence \( x_1, \ldots, x_T \) are presented sequentially to the network and used to predict output sequence \( y_1, \ldots, y_T \), or a sequence of \( T \) states that can be used for further computations. The recurrent layers have a feedback loop from the previous time step \( t - 1 \) to the current time step \( t \), which is a short-term memory also called the state. The state \( s_t \) provides the layer with information about the past and is updated at each time step \( t \). The update of the state \( s_t \) is function of the input \( x_t \), the previous state \( s_{t-1} \), and model parameters \( W \):

\[
s_t = f(x_t, s_{t-1}, W)
\]  

(2.1)
Chapter 2. Background

Figure 2.1: A recurrent unit. Left side: a recurrent unit with a feedback loop that provides it with its previous state, as indicated by the dotted arrow. Right side: unrolled version, where the state is passed on to the next time step. $f$ can be any type of activation function.

In its most basic form, $f$ looks as follows [Elman, 1990] (also see Figure 2.1):

$$f(x_t, s_{t-1}, W) = a(W^x x_t + W^s s_{t-1}) \quad (2.2)$$

Here, $a$ can be any non-linear activation function. Common choices for $a$ include tanh, which has an S-shaped curve and produces values between -1 and 1, and sigmoid, which also has an S-shaped curve but produces values between 0 and 1 and is given by $\text{sigmoid}(x) = \frac{1}{1+\exp(-x)}$ (Figure 2.2). More advanced options for $f$ include the Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] and the Gated Recurrent Unit (GRU) [Cho et al., 2014], which both use gates to modulate the flow of incoming and outgoing information.

The dimensionality of the state $s_t \in \mathbb{R}^m$ determines the amount of information that can be stored. A larger state size $m$ increases the capacity of the network, but it also increases the number of model parameters in $W^x \in \mathbb{R}^{m \times V}$ and $W^s \in \mathbb{R}^{m \times m}$. 

Figure 2.2: Left: curve of tanh with values between -1 and 1. Right: curve of sigmoid with values between 0 and 1.
\( \mathbb{R}^{m \times m} \), where \( V \) is the dimensionality of the input, which can make it more difficult to train the network. Each element of the state is also referred to as a *neuron* or a *neural unit*.

**Input**

An input sequence \( \mathbf{x}_1, ..., \mathbf{x}_T \) contains \( T \) feature vectors \( \mathbf{x}_t \in \mathbb{R}^V \). In natural language processing tasks, it is common that these feature vectors describe characters or words, referred to as types. In the bag-of-words approach, a vocabulary \( \mathbf{V} \) containing \( V \) types is constructed and used to encode the inputs. Each input \( \mathbf{x}_t \) is a vector containing \( V - 1 \) zeros, and a one in the position that corresponds to the position of the type it represents in the vocabulary \( \mathbf{V} \). For example, a vector that encodes the first type in a vocabulary has a one in its first position and zeros in all other positions. This encoding scheme is also called *one-hot encoding*.

One-hot encoded feature vectors are completely independent of each other. The distance between feature vectors is constant for every type, and remains equal for both similar and dissimilar types. However, it is usually desirable to have the feature vectors share information so that the distance between feature vectors that represent similar types becomes smaller. This can be achieved by transforming the sparse one-hot encoded vectors into dense vectors, or *embeddings*, using an embedding matrix \( \mathbf{E} \):

\[
e(\mathbf{x}_t) = \mathbf{E} \mathbf{x}_t \quad (2.3)
\]

Effectively, the one-hot encoded feature vector \( \mathbf{x}_t \) selects a single row from the embedding matrix \( \mathbf{E} \). The dimensionality of the embedding matrix \( \mathbf{E} \in \mathbb{R}^{E \times V} \) determines the dimensionality, and thus the representational capacity, of the resulting embeddings \( e(\mathbf{x}_t) \in \mathbb{R}^E \).

The embeddings matrix \( \mathbf{E} \) is a continuous representation of types in the vocabulary. [Mikolov et al., 2013b, Mikolov et al., 2013a] propose *word2vec*: a method for computing word embeddings. It uses a neural network that processes a large dataset and produces dense feature vectors in a high-dimensional space. In most current neural machine translation methods the embeddings matrix \( \mathbf{E} \) are typically treated as part of the model parameters \( \mathbf{W} \) and trained in conjunction with the rest of the parameters.

In the remainder of this section, it is assumed that \( e(\cdot) \) is a freely chosen function that may preprocess the inputs \( \mathbf{x}_t \) using an embedding matrix, but which may also simply be the identity function.

**Output**

The state \( \mathbf{s}_t \) is used to produce output predictions \( \hat{\mathbf{y}}_t \) which model the true outputs \( \mathbf{y}_t \). In case of a multi-label classification task, \( \hat{\mathbf{y}}_t \) may be a probability distribution over \( K \) classes, produced with a softmax:

\[
\hat{\mathbf{y}}_t = \sigma(\mathbf{W} \mathbf{s}_t + \mathbf{b})
\]
\[
\hat{y}_t = \frac{\exp(o_t)}{\sum_{k=0}^{K-1} \exp(o_{t,k})}
\]

\[
o_t = W^o s_t
\]

Here, \( W^o \in \mathbb{R}^{K \times m} \) maps the state \( s_t \) to log-probabilities \( o_t \) over \( K \) output classes. The predicted class \( c_t \) is then given by the class with the highest probability:

\[
c_t = \arg\max_k \hat{y}_{t,k}
\]

In many natural language processing tasks, and specifically in machine translation tasks, each class may represent a type in the output vocabulary containing \( K \) types.

**Layers**

Besides increasing the size of the state, additional capacity can be added to the network by stacking layers. By stacking \( L \) layers on top of each other, the input passes through \( L \) non-linear transformations, each governed by its own set of parameters \( W^l \). For such multi-layer networks, the state of each layer \( l \) is computed as follows:

\[
s^l_t = f^l(s^{l-1}_t, s^{l-1}_{t-1}, W^l)
\]

\[
s^0_t = x(x_t)
\]

Thus, each layer receives as input the state of the previous layer, where the first layer receives as input the original input. The state of the final layer \( s^L_t \) is used to compute the output of the network \( \hat{y}_t \).

**Training**

The effectiveness of a neural architecture depends directly on how its parameters \( W \) are chosen. When a network contains many parameters, finding the optimal parameters is not a trivial task. Typically, the parameters are initialized randomly and then updated iteratively in a way that minimizes the error (or loss) on a training dataset that consists of pairs of inputs and desired outputs. *Stochastic gradient descent* (SGD) [Bottou, 2010] is a commonly used optimization method that iteratively presents data points to the network, computes the gradient of the loss \( \mathcal{L} \) with respect to the model parameters \( W \), and then updates the model parameters \( W \) in the direction of the gradient \( \nabla_W \mathcal{L} \), scaled by the learning rate \( \eta \). The loss is computed by comparing the network prediction \( \hat{y}_t \) to the desired target \( y_t \) and should be minimized. It can have many different forms, depending on the nature of the task at hand.
2.1. Recurrent Neural Networks

SGD updates the model parameters after each data point, which consists of an input sequence \( x_1, \ldots, x_T \) and desired target sequence \( y_1, \ldots, y_T \). The update looks as follows:

\[
W \leftarrow W + \eta \nabla W \mathcal{L}
\]  

(2.7)

The model parameters \( W \) are updated in the direction that minimizes the loss \( \mathcal{L} \), scaled by a learning rate \( \eta \), which is a hyperparameter typically chosen in the range \([0.0001, 1.0]\). This update is repeated for every data point in the dataset and possibly multiple times for each data point, until a fixed number of updates have been performed or the loss \( \mathcal{L}(\hat{y}_t, y_t) \) falls below a predefined threshold. SGD is not guaranteed to find a global optimum, but with a sufficiently small learning rate that decreases over time it will find a local optimum [Bottou, 2010].

More advanced optimization methods based on SGD include Adam [Kingma and Ba, 2014], AdaDelta [Zeiler, 2012], AdaGrad [Duchi et al., 2011], and RMSGP [Tieleman and Hinton, 2012].

Loss function

The choice of loss function \( \mathcal{L} \) depends on the task at hand and directly influences how well the neural network will be able to learn and generalize. For regression tasks a common choice for the loss function is mean squared error (MSE). Given a dataset with \( N \) input and desired output pairs \((x_n, y_n)\) and model predictions \( \hat{y}_n \), the mean squared error is defined as follows:

\[
\mathcal{L}^\text{MSE} = \frac{1}{N} \sum_{n=1}^{N} (y_n - \hat{y}_n)^2
\]  

(2.8)

Most natural language processing tasks are (sequential) multi-label classification tasks, where the outputs of the model are probability distributions over classes. For such tasks a common choice for the loss function is categorical cross-entropy (CCE):

\[
\mathcal{L}^\text{CCE} = -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} y_{n,k} \log(\hat{y}_{n,k})
\]  

(2.9)

With this loss function, it is assumed that \( \hat{y}_t \) is a probability distribution over classes. Categorical cross-entropy attempts to minimize the difference between the predicted target distribution \( \hat{y}_n \) and the desired target distribution \( y_n \).

Both mean squared error and categorical cross-entropy loss are always non-negative and indicate a better fit of the model to the data as their values decrease.
Backpropagation

The parameters of the model $W$ are updated by moving them in the direction of the gradient of the loss $L$ with respect to $W$. For neural networks, this is called backpropagation, and in the case of recurrent neural networks backpropagation through time [Werbos, 1990]. The computation of the gradients makes heavy use of the chain rule for differentiation. The computation is straight-forward for the parameters $W^o$ of the output layer $o_t$ at time step $t$:

$$\nabla_{W^o} L_t = (\nabla_{\hat{y}_t} L_t) \times (\nabla_{W^o} \hat{y}_t)$$

$$= (\nabla_{\hat{y}_t} L_t) \times (\nabla_{o_t} \hat{y}_t) \times (\nabla_{W^o} o_t) \quad (2.10)$$

However, the computation of the gradient with respect to the weights of the recurrent layer $W^s$ is more complicated:

$$\nabla_{W^s} L_t = (\nabla_{\hat{y}_t} L_t) \times (\nabla_{W^s} \hat{y}_t)$$

$$= (\nabla_{\hat{y}_t} L_t) \times (\nabla_{o_t} \hat{y}_t) \times (\nabla_{W^s} o_t)$$

$$= (\nabla_{\hat{y}_t} L_t) \times (\nabla_{o_t} \hat{y}_t) \times (\nabla_{s_t} o_t) \times (\nabla_{W^s} s_t) \quad (2.11)$$

Here, complications arise because the gradient of $s_t$ with respect to $W^s$ depends on $s_{t-1}$, which depends on $s_{t-2}$, and so on. Each of these time steps depend on $W^s$. Due to this recursion, many applications of the chain rule are necessary in order to compute the gradient. In backpropagation through time, the gradients are computed by summing them over all time steps $t$ (or in order to speed up computation at the cost of precision, only over the most recent time steps):

$$\nabla_W L = \sum_{t=0}^{T} \nabla_W L_t \quad (2.12)$$

In other words, the computation of the gradient and thus the update of the model parameters is based on the contribution of each individual time step to the total loss. This becomes especially clear when the recurrent neural network is fully unrolled, because the unrolled recurrent neural network corresponds to a feed-forward neural network where the weights are shared between layers.

Unstable Gradients

The computation of the gradients involves many applications of the chain rule, which results in a product of many factors. The deeper the network or the longer the input sequences, the more factors in the product. If one of these factors is especially small or big, it will be amplified by the full product. For this reason, the gradients are inherently unstable, especially at the early layers, for which deeper products are computed. If the factors of the product are numbers smaller than 1, the final result will exponentially decrease to 0. Conversely, if these factors are numbers greater than 1, the final result will exponentially increase to infinity. The
first effect is called the vanishing gradient [Hochreiter and Schmidhuber, 1997], and the second effect is called the exploding gradient [Pascanu et al., 2013]. Both the vanishing gradient and the exploding gradient can make it difficult, if not impossible, to learn long-range dependencies in an input sequence [Bengio et al., 1994].

Whether or not the gradients become unstable depends on the depth of the model and the length of the input sequences, the values of the model parameters, and the choice of activation functions. The next paragraph will provide a rough intuition of what causes it.

Common activation functions such as tanh and sigmoid have gradients in the ranges [0, 1] and [0, 0.25], respectively (Figure 2.3). The vanishing gradient occurs when these activation functions receive values at their tails, where their outputs become constant and their gradients approach 0, causing them to become saturated. This can happen when either the inputs or the model parameters contain exceedingly small or large values. The result is that the computation of the gradients potentially involves many multiplications of numbers that approach 0, causing the gradients to exponentially decrease, or vanish. On the other hand, if the model parameters have very large values, but don’t saturate the activation functions, the gradients will also have large values. If the values are greater than 1, the computation of the gradient will potentially involve many multiplications of numbers greater than 1, and cause the gradients to exponentially grow, or explode.

The vanishing gradient problem is not easily solved. Popular solutions applicable to recurrent neural networks include using Long Short-Term Memory [Hochreiter and Schmidhuber, 1997] (LSTM) or Gated Recurrent Units (GRUs) [Cho et al., 2014], which tend not to suffer from this problem as much as standard activation functions. LSTMs and GRUs have a memory cell that is updated using only linear operations, so they have an almost constant gradient.

Figure 2.3: Left: curve of the gradient of tanh. Right: curve of the gradient of sigmoid.
Chapter 2. Background

The exploding gradient is more easily solved. The simplest solution is clipping the values of the gradients at a fixed threshold [Mikolov, 2012]. This approach does change the direction of the gradients, because the relative change of each value may not be constant, which is generally undesirable. A better solution is to clip the gradients by their $L^2$ norm [Pascanu et al., 2013], in which case the gradients are normalized such that their $L^2$ norm does not exceed a fixed threshold. Another option is to use $L^2$ regularization, a modification of the loss function that punishes having large weights.

Regularization

Ideally, a network should be able to generalize well to unseen data after training it on training data. However, at a certain point in time during training, the network may actually start generalizing less well to unseen data, and this usually happens before the loss function converges. When this happens, the network is learning an exact fit to the training data, which includes noise that may not be present in unseen data. This phenomenon is called overfitting and is generally undesirable, because most networks are trained with the objective to generalize well.

There are a number of regularization methods to prevent overfitting. The simplest method is **early stopping**: when the generalization performance on held-out validation data starts to decrease, the training phase is simply stopped. The validation data contain data points that are not present in the training data.

$L^2$ regularization is another method that aims to regularize the model parameters by making it more attractive to have smaller values. It works by adding a term to the loss function that penalizes large model parameters:

$$L^{L^2} = L + \frac{\lambda}{2}||W||^2$$  \hspace{1cm} (2.13)

With this modification of the loss function, a larger $L^2$ norm of the model parameters will induce a larger loss. This means that the optimization problem becomes a trade-off between having a small loss and having small parameter values. The $\lambda$ parameter determines the importance of having small parameter values relative to having small loss. The rough intuition behind $L^2$ regularization is that it encourages a simpler combination of parameter values, reducing the chance of overfitting.

**Dropout** [Srivastava et al., 2014] is a method that works by randomly setting parts of the outputs of the layers in the network to zero during training. At each time step, different parts of the outputs are dropped, and so effectively a different network is trained at each time step. The advantage of this method is that it prevents co-adaptation of neurons, making the full network more robust by adding redundancy. When making predictions (from unseen data) dropout is disabled, allowing the network to use its full capacity. Effectively, this full network consists
2.2. Long Short-Term Memory

Long short-term memories (LSTMs), originally proposed by [Hochreiter and Schmidhuber, 1997], are a type of recurrent neural units designed to avoid the problem of vanishing gradients, making them effective at capturing long-range dependencies in input sequences. There are many variants of the LSTM, but the version described in this section follows [Graves et al., 2013].

LSTMs consist of two main components: the hidden state and the memory vector. The memory is used to store information about the input sequence across multiple time steps. This information cannot flow freely to and from the memory, but is regulated by gates. Gates are functions that determine which parts of a vector may flow through by multiplying each element with a number between 0 and 1, typically the output of a smooth function such as the sigmoid. LSTMs have three such gates: the input gate regulates how much of the input may be written to the memory, the forget gate regulates how much of the memory should be dropped, and the output gate regulates how much of the memory should be passed on as output. See Figure 2.4 for a graphical representation.

An LSTM receives as input at time step $t$ a vector $\mathbf{x}_t$ which is the concatenation of the data input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$:

![Figure 2.4: Schematic view of Long Short-Term Memory. The square boxes correspond to the forget (f), input (i), output (o) gates and candidate memory (m'). The circles denote concatenation (·), multiplication (×) and addition (+).](image-url)
\[ H_t = \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} \] (2.14)

Using the concatenated input vector \( H_t \), a candidate memory vector \( m'_t \) is computed as a tanh transformation of the concatenated input vector \( H_t \) multiplied with weights matrix \( W^c \):

\[ m'_t = \tanh(W^c H_t) \] (2.15)

The memory is updated by simultaneously dropping parts of the previous memory \( m_{t-1} \) and adding a candidate memory \( m'_t \) to it. The forget gate \( g^f_t \) determines which parts are dropped from the previous memory \( m_{t-1} \) and the input gate \( g^i_t \) determines which parts of the input are written to the candidate memory \( m'_t \). Both gates use sigmoids, \( \circ \) denotes element-wise multiplication.

\[ m_t = g^f_t \odot m_{t-1} + g^i_t \odot m'_t \]
\[ g^f_t = \sigma(W^f H_t) \]
\[ g^i_t = \sigma(W^i H_t) \] (2.16)

The output of an LSTM is computed by applying a non-linear transformation to the memory and regulating the result with an output gate. It is computed as follows:

\[ h_t = g^o_t \odot \tanh(m_t) \]
\[ g^o_t = \sigma(W^o H_t) \] (2.17)

The output gate \( g^o_t \) is a sigmoid which produces values between 0 and 1. It determines which parts of the updated memory \( m_t \) are kept in the final output.

The initial hidden state \( h_t \) and memory vector \( m_t \) are parameters that are learned with the rest of the model parameters. Alternatively, they can be initialized with zeros at time step \( t = 0 \). The gates of the LSTM, and specifically the forget gate, prevent the gradients from vanishing. However, LSTMs are still susceptible to exploding gradients.

The forget gate was not part of the original LSTM, but was later introduced by [Gers et al., 2000]. It was found to be a crucial component by [Greff et al., 2017].

### 2.3 Encoder-Decoder Networks

The simple recurrent neural networks described Section 2.1 are capable of reading input sequences of length \( T \) and producing output sequences of the same length \( T \), but in their basic form they are limited in their ability to produce variable-length output sequences. Moreover, each output \( y_t \) depends only on inputs seen thus
far, \( x_1, \ldots, x_t \), and not on future inputs \( x_{t+1}, \ldots, x_T \). In many sequence-to-sequence modeling tasks including machine translation, these properties are shortcomings. First, the length of the desired output sequence may not necessarily be identical to the length of the input sequence. Second, simple recurrent neural networks will fail to fully capture reordering patterns. For example, a model tasked with reversing input sequences will first have to see the entire input sequence before it can start producing the reversed output sequence. For these reasons, a more flexible architecture is necessary for machine translation.

In machine translation the goal is to map an input sentence \( x_1, \ldots, x_S \) to an output sentence \( y_1, \ldots, y_T \), where \( S \) is the length of the input (source) sentence, and \( T \) is the length of the output (target) sentence. A machine translation model must be able to deal with varying sentence lengths and reordering patterns. Such a model approximates the conditional probability of a target sentence \( y = y_1, \ldots, y_T \) given input sentence \( x = x_1, \ldots, x_S \) as follows:

\[
p(y|x) = \prod_{t=1}^{T} p(y_t|x, y_1, \ldots, y_{t-1})
\]  

(2.18)

Encoder-decoder networks were designed to solve the problem of mapping sequences to sequences [Sutskever et al., 2014, Cho et al., 2014]. The encoder, typically a recurrent neural network with LSTM or GRU units, first reads the entire input sentence \( x_1, \ldots, x_S \). At each encoding step \( s \) a state \( s_{\text{enc}}^s \) is computed:

\[
s_{\text{enc}}^s = f(x_s, s_{\text{enc}}^{s-1}, W_{\text{enc}})
\]  

(2.19)

Here, \( f \) is an activation function such as LSTM or GRU and \( W_{\text{enc}} \) are the model parameters. Each state \( s_{\text{enc}}^s \) depends on the previous states \( s_{\text{enc}}^1, \ldots, s_{\text{enc}}^{s-1} \) and the current input \( x_s \). The \( S \) states are used by the decoder to translate the source sentence.

In the simplest type of encoder-decoder network, it is assumed that the final state \( s_{\text{enc}}^S \) contains all relevant information about the source sentence. The final state \( s_{\text{enc}}^S \) becomes the context vector \( c \) and the other encodings \( s_{\text{enc}}^1, \ldots, s_{\text{enc}}^{S-1} \) are discarded. In more advanced architectures, the context vector \( c_t \) is updated at each decoding step \( t \) and is be computed using all encodings \( s_{\text{enc}}^1, \ldots, s_{\text{enc}}^S \) (such as in attention models, see Section 2.3). We will refer to the context vector as \( c_t \) with the decoding step subscript \( t \) even though in the simple case the context vector may be constant for all decoding steps \( t \).

In the simple encoder-decoder network, the decoder is another recurrent neural network and its state is initialized with the context vector that represents the input sentence: \( s_{\text{dec}}^0 = c_0 \). A special token that marks the beginning of the output sentence is presented to the decoder, after which it will compute the first decoder state \( s_{\text{dec}}^1 \). The decoder state \( s_{\text{dec}}^1 \) is transformed into a prediction \( \hat{y}_1 \). Subsequent steps \( t \) depend on the previous state \( s_{\text{dec}}^{t-1} \) and take as input the previous prediction \( \hat{y}_{t-1} \). The states are computed as a non-linear transformation of the input \( \hat{y}_{t-1} \):
Figure 2.5: Schematic view of an encoder-decoder model. The encoder (blue) first encodes the input sentence \( x_1, \ldots, x_S \) and then the decoder (back) produces output sentence \( y_1, \ldots, y_T \).

\[
s_t^{\text{dec}} = f(y(\hat{y}_{t-1}), s_{t-1}^{\text{dec}}, W^{\text{dec}})
\]
\[
\hat{y}_t = o(s_t^{\text{dec}}, W^{\text{dec}})
\] (2.20)

Again, \( f \) is an activation function such as LSTM or GRU and \( W^{\text{dec}} \) are the model parameters. \( y \) is a function that maps the prediction \( \hat{y}_t \) to a continuous vector representation (e.g., using an embedding matrix). \( o \) is another activation function (e.g., a linear transformation) that maps the decoder state to a probability distribution over the target vocabulary. The decoding process is repeated until the decoder predicts a special token marking the end of the output sentence. The encoder-decoder model is depicted in Figure 2.5.

In the more advanced case (Section 2.3, Attention Mechanism), where the context vector \( c_t \) is updated at each decoding step \( t \), the decoder state is not initialized with the context vector at \( t = 0 \). Instead, the decoder state \( s_0^{\text{dec}} \) is initialized randomly and the context vector \( c_t \) is presented to the decoder as additional input at each decoding step \( t \) along with the previous prediction \( \hat{y}_{t-1} \).

One way to achieve this is to concatenate the embedding of the previous prediction \( y(\hat{y}_{t-1}) \) with the context vector \( c_t \). The update of the state \( s_t^{\text{dec}} \) becomes as follows:

\[
s_t^{\text{dec}} = f([y(\hat{y}_{t-1}); c_t], s_{t-1}^{\text{dec}}, W^{\text{dec}})
\] (2.21)

Here, \([a;b]\) denotes vector concatenation.

Note that although encoder-decoder networks consist of two separate networks, they are treated by the learning algorithm as a single network and trained in an end-to-end fashion.

**Attention Mechanism**

The simple encoder-decoder network encodes each input sentence as a fixed-size context vector \( c_t = s_S^{\text{enc}} \), independent of the length of the sentence and the current prediction. When sentences are long, this can become problematic, because more words have to be encoded into the same fixed-size vector. An attention mechanism
2.3. Encoder-Decoder Networks

Figure 2.6: Schematic view of an attention mechanism. The encoder (blue) first encodes the entire input sentence $x_1, \ldots, x_S$. At each decoding step $t$, the decoder takes a weighted sum of the encodings where $\alpha_{t,s}$ are the weights of each encoding.

can overcome this problem by selectively attending to parts of the input sentence that are relevant to the current prediction [Bahdanau et al., 2015]. Instead of using the final encoder state $s^\text{enc}_S$ as the encoding of the entire input sentence, the decoder computes a weighted average of all encodings $s^\text{enc}_1, \ldots, s^\text{enc}_S$ given its previous state $s^\text{dec}_{t-1}$ at each decoding step $t$ (Figure 2.6). The weighted average of the encodings, also called context vector $c_t$, is computed as follows:

$$c_t = \sum_{s=1}^{S} \alpha_{t,s} s^\text{enc}_s$$

$$\alpha_{t,s} = \frac{\exp(e_{t,s})}{\sum_{s=1}^{S} \exp(e_{t,s})} \quad (2.22)$$

$$e_{t,s} = W^\text{align} \tanh(W^\text{att}[s^\text{dec}_{t-1}; s^\text{enc}_s])$$

Here, $[a; b]$ denotes vector concatenation. The scalars $\alpha_{t,s}$ are a probability distribution produced by a softmax over the tokens in the input sentence, and can intuitively be interpreted as the alignment of an input token at position $s$ to the output token at position $t$. The unnormalized probabilities $e_{t,s}$ are computed using a simple feed-forward neural network that takes as input the encoding $s^\text{enc}_s$ and the previous decoder state $s^\text{dec}_{t-1}$. $W^\text{att} \in \mathbb{R}^{m \times 2m}$, where $m$ is the size of the
state of both the encoder and the decoder. $W^{\text{align}} \in \mathbb{R}^{1 \times m}$ maps the output of the tanh transformation to a scalar.

A simplification of the attention mechanism uses a simple dot product instead of the feed-forward neural network [Luong et al., 2015]. In this case, the computation of $e_{t,s}$ becomes the dot product of the encoding $s^{\text{enc}}_s$ and the previous decoder state $s^{\text{dec}}_{t-1}$:

$$e_{t,s} = s^{\text{dec}}_{t-1} \cdot s^{\text{enc}}_s$$ (2.23)

More advanced models use multiple attention mechanisms in both the encoder and the decoder, including self-attention [Vaswani et al., 2017]. Such models allow each layer in the encoder and the decoder to attend to different parts of the previous layer at each step, in addition to attending to the source encodings.

**Bi-directional Encoding**

Each encoding $s^{\text{enc}}_s$ depends on the input tokens $x_1, ..., x_s$, with a strong focus on the input token at position $s$. In other words, each encoding contains information about the corresponding input token at position $s$ and the previous tokens, but no information about future tokens. A bi-directional neural network [Schuster and Paliwal, 1997, Graves and Schmidhuber, 2005] processes the input sentence in both forward and backward order in parallel. It produces richer encodings that contain information about both the future and the past. The bi-directional encoder consists of two independent recurrent neural networks, of which one reads the input sentence in forward order and the other in backward order. The forward and backward recurrent neural networks share the embedding matrix but otherwise have their own parameters. The forward recurrent neural network produces forward encodings $\overrightarrow{s^{\text{enc}}}_s$ and the backward recurrent neural network produces backward encodings $\overleftarrow{s^{\text{enc}}}_s$. These forward and backward encodings can then be combined into bi-directional encodings $s^{\text{enc}}_s$. Possible ways of combining the forward and backward encodings include adding them together ($s^{\text{enc}}_s = \overrightarrow{s^{\text{enc}}}_s + \overleftarrow{s^{\text{enc}}}_s$) and concatenating them ($s^{\text{enc}}_s = [\overrightarrow{s^{\text{enc}}}_s, \overleftarrow{s^{\text{enc}}}_s]$). See Figure 2.7 for a graphical representation.

**2.4 Grid Long Short-Term Memory**

The $N$-dimensional Grid LSTM, proposed by [Kalchbrenner et al., 2015], is a generalization of the standard LSTM to multiple dimensions, inspired by the multi-dimensional LSTM proposed by [Graves et al., 2007]. This generalization makes it possible to arrange the units in an $N$-dimensional grid, in which each unit receives inputs on $N$ sides and generates outputs on $N$ sides. Each Grid LSTM has $N$ hidden states $h^1_t, ..., h^N_t$ and $N$ memory vectors $m^1_t, ..., m^N_t$. Unlike standard LSTMs, where each unit receives one input and produces one output
2.4. Grid Long Short-Term Memory

Figure 2.7: Schematic view of a bi-directional recurrent neural network. The inputs \( x_1, ..., x_T \) are fed to two different recurrent neural networks, of which one reads the inputs in forward order (blue) and one in backward order (black). The output states of each recurrent neural network are concatenated to produce final outputs \( s_t \).

\( h_t \), in Grid LSTMs the memory vectors \( m^1_t, ..., m^N_t \) are also part of the input and output.

The hidden states \( h^a_t \) and memory vectors \( m^a_t \) are initialized by the input vector \( x_t \) by mapping \( x_t \) into two vectors using two weights matrices with dimensions \( d \times m \) where \( d \) is the dimensionality of the inputs \( x_t \) and \( m \) is the size of the hidden states \( h^a_t \) and memory vectors \( m^a_t \).

Next, all hidden states are concatenated into a single hidden state \( H \) which is shared across all dimensions, unlike the memory vectors \( m^a_t \), which are unique to each dimension.

\[
H_t = \begin{bmatrix} h^1_t \\ \vdots \\ h^N_t \end{bmatrix}
\]  \hspace{1cm} (2.24)

A standard LSTM transformation (Equations (2.17, 2.16)) is performed for each dimension \( n \in 1...N \):

\[
(h^1_t, m^1_t) = LSTM(H_{t-1}, m^1_{t-1}, W^1) \\
\vdots \\
(h^N_t, m^N_t) = LSTM(H_{t-1}, m^N_{t-1}, W^N)
\]  \hspace{1cm} (2.25)

Here, \( W^n \) are the weight matrices of each dimension, which can potentially be shared. The output consists \( N \) hidden vectors \( h^a_t \) and memory vectors \( m^a_t \). See Figure 2.8 for a graphical representation of Grid LSTM.
Figure 2.8: Schematic view of Grid Long Short-Term Memory. Left: traditional LSTM. Middle: one-dimensional Grid LSTM (note that the one-dimensional case has no temporal dimension). Right: two-dimensional Grid LSTM, where the solid lines belong to the first dimension and the dotted lines belong to the second dimension.

**Dimension Prioritization**

The computations in Equation (2.25) are independent and performed in parallel. The value of $H_t$ is computed once and used in all $N$ LSTM transformations. However, it is possible to prioritize a specific dimension. Prioritizing dimension $n$ means that first the $N - 1$ non-prioritized LSTM transformations are computed, then the value of $H_t$ is updated with $N - 1$ updated hidden states, and then the LSTM transformation for the prioritized dimension $n$ is computed using the updated $H_t$.

For example, when prioritizing the first dimension, for that dimension Equation (2.24) changes to:

$$H = \begin{bmatrix} h_{t-1}^1 \\ h_t^1 \\ \vdots \\ h_t^N \end{bmatrix}$$

(2.26)

Dimension prioritization can be especially useful for output dimensions.

**Non-LSTM Dimensions**

Grid LSTMs do not necessarily have LSTM transformations in each dimension. It is also possible to have regular connections. For dimensions with regular connections, the transformation in Equation (2.25) can simply be replaced by a non-linear activation function. This looks as follows for the first dimension:

$$h_t^1 = a(H_{t-1}, W^1)$$

(2.27)
Here, $a$ can be any non-linear activation function. Note that such regular dimensions receive just one input $h_n$ and produce one output $h'_n$.

### Comparison to standard LSTM

Grid LSTM is a generalization of the standard LSTM, meaning that any LSTM network can be modeled by a Grid LSTM network. For example, the standard LSTM used in sequence-to-sequence learning is equivalent to a two-dimensional Grid LSTM with LSTM connections in the temporal dimension, but regular identity connections in the depth dimension, where the depth dimension reads inputs and produces outputs.

### Application in Neural Machine Translation

Grid LSTM was applied by [Kalchbrenner et al., 2015] in a novel model for machine translation. This model, called *Re-encoder*, views translation as a two-dimensional mapping from source sentence to target sentence. One dimension processes the target sentence, while the other repeatedly processes the source dimension. This allows it to encode and attend to the source sentence differently depending on where it is in the translation process. This model is extended in this thesis and fully explained in Section 3.1.

### 2.5 Active Memory & Re-encoding

The attention mechanism described in Section 2.3 is responsible for a large part of the success of neural networks in machine translation [Bahdanau et al., 2015], as well as in other domains such as image recognition and captioning [Xu et al., 2015] and in Neural Turing Machines that are able to learn arbitrary algorithmic tasks [Graves et al., 2014].

Models with an attention mechanism operate on their memory by attending to specific parts relevant to the next step. In the case of an encoder-decoder model, the memory consists of the sequence of source encodings. To a limited extent, the encoder-decoder model can manipulate its memory by recombining the source encodings in different ways. However, the attention mechanism for encoder-decoder models is constrained by the use of a softmax function over the source encodings [Kaiser and Bengio, 2016]. The softmax tends to assign most probability mass to a single item, so the model tends to focus its attention on a single item in memory. This is an undesirable effect in many cases, where the model may need to attend to multiple items simultaneously.

According to [Kaiser and Bengio, 2016] and as previously shown in [Kaiser and Sutskever, 2015], this problem can be overcome by allowing the model to access and manipulate its memory at each decoding step, using what they call an *active memory*. The Re-encoder model by [Kalchbrenner et al., 2015], briefly
introduced in Section 2.4 and further explained in Section 3.1, is to a large extent an active memory model because it re-encodes the entire source sentence at each decoding step.

The standard attention mechanism is to a limited extent an active memory. However, it is possible to extend the attention mechanism so that is able to freely manipulate the memory by allowing it to re-encode the source encodings at each decoding step depending on the decoder state. This idea, which is similar to [Zhang et al., 2017], is proposed and explained in detail in Section 3.2.

An important part of this thesis consists of investigating to which degree re-encoding can improve the performance of neural models for machine translation.
Chapter 3
Models

This chapter introduces two neural machine translation models. Both models make heavy use of Grid LSTM units, a generalization of LSTMs to multiple dimensions. The first model, called the Grid Re-encoder, is based on the translation model proposed by [Kalchbrenner et al., 2015] that processes the target sentence in one dimension and the source sentence in another, repeatedly re-encoding the source sentence for each decoding step. The second model, called the Grid Encoder-Decoder, is based on the encoder-decoder architecture and has an attention mechanism that can manipulate the encodings based on the decoder state.

3.1 Model I: Grid Re-encoder

The Re-encoder model, proposed by [Kalchbrenner et al., 2015], views translation as a two-dimensional mapping from source sentence to target sentence. It is a network consisting of two two-dimensional grids of size $T \times S$, where $T$ is the length of the target sentence and $S$ is the length of the source sentence. The first (target) dimension predicts the target sentence while the second (source) dimension repeatedly encodes the source sentence for each target prediction. The two grids are placed on top of each other and connected at each position in a third, intermediate dimension. The first two dimensions use Grid LSTM connections, the intermediate dimension uses identity connections.

For each target word, the model first reads the entire source sentence in forward order in the top, forward grid. Next, the model reads the entire source sentence in backward order in the bottom, backward grid. At each position in the grid, the backward grid receives an input from the forward grid in the intermediate dimension, so that information from the forward grid can flow to the backward grid. At each step, the output in each dimension is passed on to the next step and used as input in that dimension. See Figure 3.1 for a depiction of the model.
Figure 3.1: The Grid Re-encoder model. Each box represents a Grid LSTM and shares its parameters with Grid LSTMs represented by boxes of the same color. The black boxes represent the forward grid, the blue boxes represent the backward grid. At each position in the grid, there is an identity connection from the forward grid to the backward grid.

The final target prediction is given by the output of the target dimension of the backward grid. Since the source sentence is re-encoded for each prediction, the model can (implicitly) attend to different parts of the source sentence depending on previous predictions.

A more advanced version with an explicit attention mechanism is also proposed. For this model, the final outputs come from both the target dimension and the source dimension. An attention mechanism is placed over the source words and combines the target and source encodings into a single output by concatenating and summing them using a weighted sum. Figure 3.2 depicts a single prediction step for this model.

3.1.1 Re-encoder

For each target prediction, the model scans every word in the source sentence and computes the updates for the target, source and intermediate dimensions (this corresponds to a pass through one column in Figure 3.1). The target and source inputs are first processed by the forward grid, which receives inputs in the target and source dimensions, but not in the intermediate dimension. The inputs of the target and source dimensions are hidden state and memory vector pairs, given by:
Figure 3.2: The Grid Re-encoder model with attention for a single target prediction. The $x_{t,s}$ correspond to the source words at $t = 0$ or the previous outputs in the source dimension at $t - 1$.

\[
\begin{align*}
\vec{t}_{t,s} &= [\vec{h}_{t,s}^{\text{target}}, \vec{m}_{t,s}^{\text{target}}] \\
\vec{s}_{t,s} &= [\vec{h}_{t,s}^{\text{source}}, \vec{m}_{t,s}^{\text{source}}] \\
\vec{H}_{t,s} &= [\vec{h}_{t,s}^{\text{target}}, \vec{h}_{t-1,s}^{\text{source}}]
\end{align*}
\] (3.1)

Here, the $t$-subscript indicates positions in the target sentence (on the horizontal axis in Figure 3.1), and the $s$-subscript indicates positions in the source sentence (on the vertical axis). With these inputs, the forward grid computes a three-dimensional Grid LSTM transformation for each word in the source sentence. The target and source dimensions use LSTM connections and the intermediate dimension uses identity connections. The outputs of the three dimensions are computed as follows:

\[
\begin{align*}
\vec{t}_{t,s} &= \text{LSTM}(\vec{H}_{t,s}, \vec{m}_{t,s}^{\text{target}}, \vec{W}^{\text{target}}) \\
\vec{s}_{t,s} &= \text{LSTM}(\vec{H}_{t,s}, \vec{m}_{t-1,s}^{\text{source}}, \vec{W}^{\text{source}}) \\
\vec{d}_{t,s} &= \vec{W}^{\text{intermediate}} \vec{H}_{t,s}
\end{align*}
\] (3.2)

At each position in the grid, the input to the target dimension (which corresponds to the vertical columns in Figure 3.1) is given by the output of cell above at position $(t, s - 1)$; the input to the source dimension (horizontal rows) is given by the output of the cell to the left at position $(t - 1, s)$. The third dimension has regular identity connections and receives no input, but does produce an output $\vec{h}_{t,s}^{\text{intermediate}}$ which is given as input to the corresponding cell in the backward
grid.

The backward grid reads the source sentence in reverse order and computes almost identical LSTM transformations. The backward grid receives an input from the corresponding position of the forward grid at each position in the intermediate dimension. The inputs for the backward grid look as follows:

\[
\begin{align*}
\hat{t}_{t,s} &= \begin{bmatrix} \hat{h}_{t,s}^{\text{target}} ; \hat{m}_{t,s}^{\text{target}} \end{bmatrix} \\
\hat{s}_{t,s} &= \begin{bmatrix} \hat{h}_{t,s}^{\text{source}} ; \hat{m}_{t,s}^{\text{source}} \end{bmatrix} \\
\hat{H}_{t,s} &= \begin{bmatrix} \hat{h}_{t,s}^{\text{target}} - 1 ; \hat{h}_{t-1,s}^{\text{source}} ; \hat{d}_{t,S-s+1}^{\text{intermediate}} \end{bmatrix}
\end{align*}
\] (3.3)

Like the forward grid, the backward grid performs three-dimensional Grid LSTM transformations for each word in the source sentence. Again, the target and source dimensions use LSTM connections and the intermediate dimension uses identity connections. The outputs of the backward grid look as follows:

\[
\begin{align*}
\hat{t}_{t,s} &= \text{LSTM}(\hat{H}_{t,s}; \hat{m}_{t,s}^{\text{target}} ; \hat{W}^{\text{target}}) \\
\hat{s}_{t,s} &= \text{LSTM}(\hat{H}_{t,s}; \hat{m}_{t,s}^{\text{source}} ; \hat{W}^{\text{source}}) \\
\hat{d}_{t,s} &= \hat{W}^{\text{intermediate}} \hat{H}_{t,s}
\end{align*}
\] (3.4)

Since information from the forward grid can flow to the backward grid at each position in the grid, the outputs of the backward grid contain information about both future and past. The outputs of the backward grid are bi-directional encodings and can be used for decoding.

At \(t = 0\) (before any predictions are done, so the input to the left-most column in Figure 3.1), the hidden states and memory vectors of the source dimension are initialized using the source words \(x_1, \ldots, x_S\). At \(s = 0\) (before the source sentence is read, so the input to the top-most row), the hidden state and memory vector of the target dimension in the forward grid is initialized using the previous target prediction \(y_{t-1}\). In the backward grid, the target dimension is initialized with the output in the target dimension of the forward grid.

To initialize the hidden states and memory vectors with words, the one-hot encoded input words are multiplied with an embedding matrix \(E\) to obtain word embeddings, which are then mapped to a hidden state and memory vector pair using a weights matrix \(I\).

\[
\begin{align*}
\hat{t}_{t,0} &= \hat{I}^{\text{target}} E^{\text{target}} y_{t-1} \\
\hat{t}_{t,0} &= \hat{t}_{t,S} \\
\hat{s}_{0,s} &= \hat{I}^{\text{source}} E^{\text{source}} x_s \\
\hat{s}_{0,s} &= \hat{s}_{0,s-1+1}
\end{align*}
\] (3.5)

Here, \(x_s\) and \(y_t\) are the one-hot encoded source and target input words. When training \(y_s\) are target labels, when testing they are generated target pre-
dictions. The first target input $y_0$ is a special beginning-of-sentence token. $E^{\text{target}} \in \mathbb{R}^{e_{\text{target}} \times v_{\text{target}}}$ is the embedding matrix, where $e_{\text{target}}$ is the embedding size and $v_{\text{target}}$ the target vocabulary size. Both forward and backward $I^{\text{target}} \in \mathbb{R}^{2m \times e_{\text{target}}}$ contain weights that are used to initialize the hidden state and memory vector of the cell. Analogous definitions apply to the initialization of the source dimension. Note that the embedding matrices $E$ are shared by the forward and backward grids.

To predict target words, the final outputs of the backward grid are used by the decoder:

$$
\begin{align*}
    t_{t,s} &= \overrightarrow{t}_{t,s} \\
    s_{t,s} &= \overleftarrow{s}_{t,s} \\
    d_{t,s} &= \overleftarrow{d}_{t,s}
\end{align*}
$$

(3.6)

See Algorithm 1 for the re-encoding procedure for a single target prediction.

### 3.1.2 Decoder

The Grid Re-encoder model does not have a dedicated decoder like the standard encoder-decoder models. Instead, encoding and decoding is done simultaneously and repeatedly. The Grid Re-encoder directly predicts the target words using the outputs of the backward layer. The outputs consist of $S$ target outputs $t_{t,s}$, source outputs $s_{t,s}$ and intermediate outputs $d_{t,s}$. We propose two ways to use these outputs for decoding:

1. Simple decoding: the first variant follows [Kalchbrenner et al., 2015] and computes a probability distribution over the target vocabulary using only the final output of the target dimension.

2. Decoding with attention: the second variant makes use of an attention mechanism [Bahdanau et al., 2015] over the outputs of the source dimension (one for each source word).

**Simple Decoding**

With simple decoding, the model simply uses the final output of the target dimension for decoding. In this case, the model must learn to propagate all relevant information for prediction to this final output. The decoding is given by:

$$o_t = t_{t,S}$$

(3.7)
Decoding with Attention

This variant is not limited to the final target output of the backward grid. Instead, it uses both the target and source outputs of the backward grid at all source positions $s$. At each source position $s$, the target and source output are concatenated and used to compute a weighted average. An attention mechanism based on [Bahdanau et al., 2015] computes the relevance of each source position to the next target prediction and produces a weighted sum of the target and source outputs. The relevance of each source position $s$ is computed by feeding the outputs of the intermediate dimension of position to a simple feed-forward neural network. The intermediate output $d_{t,s}$ contains information about the current target word and the source word at position $s$. The attention mechanism is depicted in Figure 3.2.

\[
\alpha_{t,s} = \text{softmax}(\mathbf{W}^\text{att} \tanh(\mathbf{d}_{t,s}))
\]

(3.8)

The unnormalized energies $e_{t,s}$ of the source encodings are computed using a simple feed-forward neural network, with parameters $\mathbf{W}^\text{att} \in \mathbb{R}^{1 \times m}$. The energies $e_{t,s}$ are then normalized using a softmax function, producing probabilities $\alpha_{t,s}$ that give the alignment between source encodings at position $s$ and the target word at position $t$.

The difference with [Bahdanau et al., 2015] is that this variant does not directly compare a source encoding to the previous decoder state, but uses the output of the intermediate dimension to compute the alignment instead. Other methods have been proposed that use a recurrent neural network for attention modeling [Yang et al., 2016]. The attention mechanism proposed for the Grid Re-encoder can be seen as a hybrid version of the standard attention mechanism and the recurrent attention mechanism.

Prediction

The output $o_t$ of the decoder is mapped to a prediction. The final prediction $y_t$ is a probability distribution over the target vocabulary produced by a softmax:

\[
y_t = \text{softmax}(\mathbf{W}^\text{output} o_t)
\]

(3.9)

Here, $\mathbf{W}^\text{output} \in \mathbb{R}^{v_{\text{target}} \times m'}$ (where $m' = 2m$ for simple decoding, $m' = 4m$ for decoding with attention) are trainable weights, $v_{\text{target}}$ is the target vocabulary.
3.1. Model I: Grid Re-encoder

size, $T$ and $m$ is size of the hidden state and memory vector of the cells in the encoder.

If we are only interested in the word with the highest probability, the softmax becomes a hardmax where the word with the highest probability gets a value of one and the rest gets a value of zero. This is typically the case when predicting from unseen data, during development and testing.

See Algorithm 2 for the decoding procedure for a given target position $t$. 
Algorithm 1 Grid Re-encoder: Re-encoding procedure.

1: procedure RE-ENCODE(x, y, t)
2:  // Initialize forward layer with source and target words
3: \[ \vec{t}_{t,0} \leftarrow \hat{I}_{\text{target}}(E_{\text{target}} y_{t-1}) \]
4: \[ \vec{s}_{0,s} \leftarrow \hat{I}_{\text{source}}(E_{\text{source}} x_s) \]
5: \[ \]
6:  // Initialize backward layer with source and target words
7: \[ \vec{t}_{t,0} \leftarrow \hat{I}_{\text{target}}(E_{\text{target}} y_{t-1}) \]
8: \[ \vec{s}_{0,s} \leftarrow \hat{I}_{\text{source}}(E_{\text{source}} x_{S-s+1}) \]
9: \[ \]
10: // Compute forward encodings
11: for \( s = 1..S \) do
12: \[ \vec{t}_{t,s} \leftarrow \text{LSTM}(\vec{H}_{t,s}, \vec{m}_{t,s-1}, \vec{W}_{\text{target}}) \]
13: \[ \vec{s}_{t,s} \leftarrow \text{LSTM}(\vec{H}_{t,s}, \vec{m}_{t-1,s}, \vec{W}_{\text{source}}) \]
14: \[ \vec{d}_{t,s} \leftarrow \vec{W}_{\text{intermediate}} \vec{H}_{t,s} \]
15: \[ \]
16: // Compute backward encodings
17: for \( s = 1..S \) do
18: \[ \vec{t}_{t,s} \leftarrow \text{LSTM}(\vec{H}_{t,s}, \vec{m}_{t,s-1}, \vec{W}_{\text{target}}) \]
19: \[ \vec{s}_{t,s} \leftarrow \text{LSTM}(\vec{H}_{t,s}, \vec{m}_{t-1,s}, \vec{W}_{\text{source}}) \]
20: \[ \vec{d}_{t,s} \leftarrow \vec{W}_{\text{intermediate}} \vec{H}_{t,s} \]
21: \[ \]
22: return \( \vec{t}, \vec{s}, \vec{d} \)

Algorithm 2 Grid Re-encoder: Decoding procedure.

1: procedure DECODE(x)
2: \[ y_0 \leftarrow \text{BOS} \]
3: \[ t \leftarrow 1 \]
4: \[ \]
5: while \( y_{t-1} \) is not EOS do
6: \[ t_t, s_t, d_t \leftarrow \text{RE-ENCODE(x, y, t)} \]
7: \[ \]
8: // Extract encoding: either Equation (3.7) (simple) or Equation (3.8) (attention)
9: \[ o_t \leftarrow \text{ATTEND}(t, t_t, s_t, d_t) \]
10: \[ \]
11: // Final target prediction (Equation (3.9))
12: \[ y_t \leftarrow W_{\text{output}} o_t \]
13: \[ \]
14: \[ t \leftarrow t + 1 \]
15: \[ \]
16: return \( y \)
3.2. Model II: Grid Encoder-Decoder

The Grid Encoder-Decoder is based on the encoder-decoder network with attention by [Bahdanau et al., 2015]. It uses Grid LSTM units where one dimension is responsible for processing the inputs and the other for processing the state, which captures temporal relationships. The encoder is bi-directional, so each encoding contains information about both the future and the past. An attention mechanism combines the encodings into a single context vector at each decoding step, depending on the state of the decoder. A novel addition is the active attention mechanism: before attending to the encodings, the model re-encodes the source sentence based on the current decoder state. This mechanism allows the model
to alter the encodings to make them more relevant to the next decoding step (Figure 3.3).

### 3.2.1 Bi-directional Encoder

The encoder consists of two two-dimensional Grid LSTMs, of which one reads the sentence from left to right and the other from right to left. The first dimension reads the source words and produces the encodings the second dimension processes the state which allows information to flow from one time step to another during the encoding process.

The inputs $\rightarrow \mathbf{i}_{s}^{enc}$ and states $\rightarrow \mathbf{s}_{s}^{enc}$ at position $s$ are defined as a concatenation of a hidden state and a memory vector:

$$
\rightarrow \mathbf{i}_{s}^{enc} = [\rightarrow \mathbf{h}^{enc-input}_{s}; \rightarrow \mathbf{m}^{enc-input}_{s}]
$$
$$
\rightarrow \mathbf{s}_{s}^{enc} = [\rightarrow \mathbf{h}^{enc-state}_{s}; \rightarrow \mathbf{m}^{enc-state}_{s}]
$$

(3.10)

The hidden state $\rightarrow \mathbf{h}^{enc-input}_{s}$ and memory vector $\rightarrow \mathbf{m}^{enc-input}_{s}$ of the input $\rightarrow \mathbf{i}_{s}^{enc}$ are initialized with the input word. The one-hot encoded input word $\rightarrow \mathbf{x}_{s}$ is multiplied with an embedding matrix $\mathbf{E}_{source}$, resulting in a word embedding that is then mapped into a hidden state and memory vector:

$$
\rightarrow \mathbf{h}^{enc-input}_{s} = \rightarrow \mathbf{I}^{enc-hidden}_{s} \mathbf{E}_{source} \rightarrow \mathbf{x}_{s}
$$
$$
\rightarrow \mathbf{m}^{enc-input}_{s} = \rightarrow \mathbf{I}^{enc-memory}_{s} \mathbf{E}_{source} \rightarrow \mathbf{x}_{s}
$$

(3.11)

$\mathbf{E}_{source} \in \mathbb{R}^{e_{source} \times v_{source}}$ is the embedding matrix, where $e_{source}$ is the embedding size and $v_{source}$ the vocabulary size. $\rightarrow \mathbf{I}^{enc-hidden}_{s} \in \mathbb{R}^{m \times e_{source}}$ and $\rightarrow \mathbf{I}^{enc-memory}_{s} \in \mathbb{R}^{m \times e_{source}}$ are used to initialize the hidden state and memory vector respectively, where $m$ denotes the size of the hidden state and memory vector. $\mathbf{E}_{source}$, $\rightarrow \mathbf{I}$, and $\rightarrow \mathbf{s}^{enc}_{0}$ are initialized using an initialization strategy of choice (e.g., Xavier [Glorot and Bengio, 2010]).

The forward encodings $\rightarrow \mathbf{c}_{s} = [\rightarrow \mathbf{h}^{enc-output}_{s}; \rightarrow \mathbf{m}^{enc-output}_{s}]$ and states $\rightarrow \mathbf{s}^{enc}_{s}$ are updated using a two-dimensional Grid LSTM transformation. The input is processed in one dimension, the state in the other. The update is computed as follows:

$$
\rightarrow \mathbf{c}_{s} = \text{LSTM}(\rightarrow \mathbf{h}^{enc}_{s}, \rightarrow \mathbf{m}^{enc-input}_{s}, \mathbf{W}^{enc-input})
$$
$$
\rightarrow \mathbf{s}^{enc}_{s} = \text{LSTM}(\rightarrow \mathbf{h}^{enc}_{s}, \rightarrow \mathbf{m}^{enc-state}_{s-1}, \mathbf{W}^{enc-state})
$$
$$
\rightarrow \mathbf{h}^{enc}_{s} = [\rightarrow \mathbf{i}^{enc-input}_{s}, \rightarrow \mathbf{h}^{enc-state}_{s-1}]
$$

(3.12)

The backward encodings are computed analogously. The only difference is that $\rightarrow \mathbf{s}^{enc}_{0} = \rightarrow \mathbf{s}^{enc}_{S}$, where $S$ is the length of the source sentence (i.e., the backward state is initialized by the final forward state). This is also different from
3.2. Model II: Grid Encoder-Decoder

[Bahdanau et al., 2015], where both the forward and the backward state are initialized randomly and independently. Note that the embedding matrices \( E_{\text{source}} \) are shared by the forward and backward layers.

Finally, we end up with \( S \) forward and backward encodings. The goal is to encode both future and past information into each vector, so they are combined into bi-directional encodings \( c_s \) by adding them together:

\[
\begin{align*}
    c_s &= [h_s^{\text{enc}}, m_s^{\text{enc}}] \\
    h_s^{\text{enc}} &= \overrightarrow{h_s} + \overleftarrow{h_s} \\
    m_s^{\text{enc}} &= \overrightarrow{m_s} + \overleftarrow{m_s}
\end{align*}
\]  

(3.13)

See Algorithm 3 for the encoding procedure.

3.2.2 Decoder

The decoder is a three-dimensional Grid LSTM that processes the source encodings in the first dimension, the previous prediction in the second (non-LSTM) dimension, and the decoder state in the third dimension.

The inputs at time step \( t \) are given by:

\[
\begin{align*}
    \bar{c}_t &= \text{attend}(c_1, ..., c_S, s_{t-1}^{\text{dec}}) \\
    \bar{h}_t^{\text{dec}} &= E_{\text{target}} y_{t-1} \\
    s_t^{\text{dec}} &= [h_t^{\text{dec-state}}, m_t^{\text{dec-state}}]
\end{align*}
\]  

(3.14)

Here, \( \text{attend}(\cdot) \) is a function that computes a context vector \( \bar{c}_t \) given the source encodings \( c_s \) and previous decoder state \( s_{t-1}^{\text{dec}} \). The context vector is a representation of the source sentence containing the encodings that are most relevant to the next prediction. It is an attention mechanism [Bahdanau et al., 2015] that computes a context vector \( \bar{c}_t \) based on the alignment between source encodings \( c_s \) and the previous decoder state \( s_{t-1}^{\text{dec}} \). The more relevant a source encoding \( c_s \) is to the next prediction, the more prevalent it will be in the context vector \( \bar{c}_t \).

Two variants are proposed:

1. Passive attention: the first variant computes a weighted average of the source encodings \( c_s \) based on the previous decoder state \( s_{t-1}^{\text{dec}} \).

2. Active attention: the second variant computes a weighted average and also actively updates the source encodings \( c_s \) based on the previous decoder state \( s_{t-1}^{\text{dec}} \).

Passive Attention

The passive attention mechanism follows [Bahdanau et al., 2015] and computes a weighted average of the source encodings \( c_s \). It allows the model to selectively
attend to parts of the source sentence depending on the words translated so far, without changing the representations. Each weight is computed by checking the alignment between the source encoding and the previous decoder state \( s_{\text{dec}}^{t-1} \). The encodings are multiplied with the weights and added together, producing the context vector \( c'_t, s = [h^\text{context}_t, m^\text{context}_t] \):

\[
\bar{c}_t = \sum_{s=1}^{S} \alpha_{t,s} c_s
\]

\[
\alpha_{t,s} = \frac{\exp(e_{t,s})}{\sum_{s=1}^{S} \exp(e_{t,s})}
\]

\[
e_{t,s} = W^\text{align} \tanh(W^\text{att}[s_{\text{dec-state}}^{t-1}; c_s])
\]

The unnormalized energies \( e_{t,s} \) of the source encodings are computed using a simple feed-forward neural network, with parameters \( W^\text{align} \in \mathbb{R}^{m \times 4m} \) and \( W^\text{att} \in \mathbb{R}^{1 \times m} \). The energies \( e_{t,s} \) are then normalized using a softmax probability distribution, producing probabilities \( \alpha_{t,s} \) that give the alignment between source encodings at position \( s \) and the target word at position \( t \).

### Active Attention

The active attention mechanism uses a Grid LSTM to simultaneously update the encodings \( c_s \) (producing re-encodings \( c'_t, s \)) and compute the alignment to the previous decoder state \( s_{\text{dec}}^{t-1} \). The context vector is a weighted average of the re-encodings and is computed similarly to the passive attention mechanism:

\[
\bar{c}_t = \sum_{s=1}^{S} \alpha_{t,s} c'_t, s
\]

\[
\alpha_{t,s} = \frac{\exp(e_{t,s})}{\sum_{s=1}^{S} \exp(e_{t,s})}
\]

\[
e_{t,s} = W^\text{align} a_{t,s}
\]

The re-encodings \( c'_t, s = [h^\text{context}_s, m^\text{context}_s] \) and the attention vectors \( a_{t,s} \) are the result of a two-dimensional Grid LSTM transformation. The computation takes as input the bi-directional encoding \( c_s = [h^\text{enc}_s, m^\text{enc}_s] \) and previous decoder state \( s_{\text{dec-state}}^{t-1} = [h^\text{dec-state}_s, m^\text{dec-state}_s] \). The Grid LSTM transformation outputs a re-encoding \( c'_t, s \) in one dimension and an attention vector \( a_{t,s} \) in the other.

\[
c'_t, s = \text{LSTM}(H^\text{re-enc}_s, m^\text{enc}_s, W^\text{re-enc})
\]

\[
a_{t,s} = \text{LSTM}(H^\text{re-enc}_s, m^\text{dec-state}_s, W^\text{att})
\]

\[
H^\text{re-enc}_s = [h^\text{enc}_s, h^\text{dec-state}_s]
\]

An important difference with the previously introduced Grid Re-encoder model is that at each decoding step the Grid Encoder-Decoder uses the original encod-
3.2. Model II: Grid Encoder-Decoder

ings \( c_t \) and re-encodes them. The re-encodings \( c'_t \) are only used for decoding step \( t \), and discarded after the prediction has been made. In the Grid Re-encoder, re-encodings of the source sentence are passed on to the next target prediction and re-used. Superficially, the Grid Re-encoder uses ‘deep’ re-encoding and the Grid Encoder-Decoder uses ‘shallow’ re-encoding.

Decoding

The decodings \( o_t \) and decoder states \( s^{\text{dec}}_t \) are the result of a two-dimensional Grid LSTM transformation. The computation takes as input the context vector \( \tilde{c}_t \), the decoder input \( i^\text{dec}_t \), and the previous decoder state \( s^{\text{dec-state}}_{t-1} \). It outputs a decoding \( o_t \) which is transformed to an output word and a new state \( s^{\text{dec-state}}_t \).

\[
\begin{align*}
    o_t & = \text{LSTM}(H^{\text{dec}}_t, m^\text{context}_t, W^{\text{dec-context}}) \\
    s^{\text{dec}}_t & = \text{LSTM}(H^{\text{dec}}_t, m^{\text{dec-state}}_{t-1}, W^{\text{dec-state}}) \\
    H^{\text{dec}}_t & = [h^\text{context}_t; i^\text{dec}_t; h^{\text{dec-state}}_{t-1}]
\end{align*}
\]  

(3.18)

Prediction

The output \( o_t \) of the decoder is mapped to a prediction. The final prediction \( y_t \) is a probability distribution over the target vocabulary produced by a softmax:

\[
y_t = \text{softmax}(W^{\text{output}} o_t)
\]

\[
= \frac{\exp(W^{\text{output}} o_t)}{\sum_{k=0}^{v^{\text{target}}} \exp([W^{\text{output}} o_t]_k)}
\]  

(3.19)

Here, \( W^{\text{output}} \in \mathbb{R}^{v^{\text{target}} \times 2m} \) are trainable weights, \( v^{\text{target}} \) is the target vocabulary size, and \( m \) is size of the hidden state and memory vector of the cells in the encoder.

If we are only interested in the word with the highest probability, the softmax becomes a hardmax where the word with the highest probability gets a value of one and the rest gets a value of zero. This is typically the case when predicting from unseen data, during development and testing.

See Algorithm 4 for the decoding procedure for a given target position \( t \).
Algorithm 3 Grid Encoder-Decoder: Encoding procedure

1: procedure ENCODE(x)
2:    // Compute forward encodings
3:    for s = 1..S do
4:      $\overrightarrow{c}_s \leftarrow \text{LSTM}(\overrightarrow{H}^{\text{enc}}_s, \overrightarrow{m}^{\text{enc-input}}_s, \overrightarrow{W}^{\text{enc-input}})$
5:      $\overrightarrow{s}^{\text{enc}}_s \leftarrow \text{LSTM}(\overrightarrow{H}^{\text{enc}}_s, \overrightarrow{m}^{\text{enc-state}}_{s-1}, \overrightarrow{W}^{\text{enc-state}})$
6:    
7:    // Set initial backward state to final forward state
8:      $\overleftarrow{s}^0_{\text{enc}} \leftarrow \overrightarrow{s}^S_{\text{enc}}$
9:    
10:   // Compute backward encodings
11:    for s = 1..S do
12:      $\overleftarrow{c}_s \leftarrow \text{LSTM}(\overleftarrow{H}^{\text{enc}}_s, \overleftarrow{m}^{\text{enc-input}}_s, \overleftarrow{W}^{\text{enc-input}})$
13:      $\overleftarrow{s}^{\text{enc}}_s \leftarrow \text{LSTM}(\overleftarrow{H}^{\text{enc}}_s, \overleftarrow{m}^{\text{enc-state}}_{s-1}, \overleftarrow{W}^{\text{enc-state}})$
14:    
15:    // Combine forward and backward encodings
16:    for s = 1..S do
17:      $c_s \leftarrow [h^{\text{enc}}_s, m^{\text{enc}}_s]$ // See Equation (3.13)
18:  
19:    return $c$
3.2. Model II: Grid Encoder-Decoder

Algorithm 4 Grid Encoder-Decoder: Decoding procedure

1: procedure DECODE(x)
2: \hspace{10pt} \text{c} \gets \text{ENCODE}(x)
3: \hspace{10pt} \text{y}_0 \gets \text{BOS} // Beginning-of-sentence token
4: \hspace{10pt} t \gets 1
5: \hspace{10pt} \text{while } \text{y}_{t-1} \text{ is not EOS do } // End-of-sentence token
6: \hspace{20pt} // Compute context vector: either Equation (3.15) (passive) or Equation (3.16, 3.17) (active).
7: \hspace{30pt} // It is a hidden state and memory vector pair: \[ \mathbf{h}_{t}^{\text{context}}, \mathbf{m}_{t}^{\text{context}} \].
8: \hspace{30pt} \bar{c}_t \gets \text{Attend}(\text{c}, \text{s}_{\text{dec}t})
9: \hspace{10pt} \text{// Decode target word (Equation (3.18)).}
10: \hspace{20pt} \text{// H}_{\text{dec}t} \text{ contains the previous target word, the hidden state of the context vector, and the hidden state of the previous decoder state.}
11: \hspace{30pt} \text{o}_t \gets \text{LSTM}(\text{H}_{\text{dec}t}, \mathbf{m}_{t}^{\text{context}}, \mathbf{W}_{\text{dec-context}})
12: \hspace{10pt} \text{// Predict target word (Equation (3.19)).}
13: \hspace{20pt} \text{y}_t \gets \text{softmax}(\mathbf{W}_{\text{output}} \text{o}_t)
14: \hspace{10pt} t \gets t + 1
15: \hspace{10pt} \text{return } \text{y}
Chapter 4

Experiments

In this chapter we will evaluate the Grid Re-encoder and Grid Encoder-Decoder proposed in Chapter 3 on two corpora. We will compare their performance and properties to a baseline model and other results in the literature. We will do an in-depth investigation into the contribution of attention, re-encoding and other parameters. Finally, we will do a qualitative analysis to find out where the models shine and where they fail.

4.1 Data

Chinese-to-English

The IWSLT BTEC Chinese-to-English corpus contains 44016 sentence pairs for training, 1006 for validation, and 506 for testing. The English sentences in the train data are between 3 and 66 words long, and 9 words on average. Words that only occur once are filtered out and replaced with an UNKNOWN token. After preprocessing, the Chinese vocabulary contains 7055 words and the English vocabulary 5646 words.

Figure 4.1 shows the distribution of Chinese sentence lengths of the train data, which is slightly skewed: the majority of the sentences are between 6 and 10 words long. Despite this imbalance, this corpus was chosen to allow for comparisons with the existing Re-encoder model by Kalchbrenner, and because it is relatively small, which speeds up experimentation. A representative example from the BTEC Chinese-to-English corpus looks as follows:

ZH — 去 市 政 府 的 巴 士 站 在 哪 儿 ？

EN — where is the bus stop for city hall ?
Chapter 4. Experiments

Figure 4.1: Distribution of source sentence lengths for the BTEC Chinese-to-English train sentences (left) and the ASPEC Japanese-to-English train sentences (right).

**Japanese-to-English**

ASPEC, from WAT ’16 [WAT, 2016], is a Japanese-to-English corpus containing approximately 3M sentence pairs for training, 1790 for validation, and 1812 for testing. However, only the first 100K sentences of the train data are used for training. The shortest English sentence contains 3 words, and the average is 20 words. The corpus is preprocessed following the directions given by [WAT, 2016]. After that, all words that occur fewer than two times are replaced by the **UNKNOWN** token. The resulting Japanese vocabulary contains 17329 words, the English vocabulary contains 21099 words.

Figure 4.1 shows the distribution of Japanese sentence lengths of the train data. It shows that this corpus consists of a wider range of sentence lengths and is more balanced. The ASPEC Japanese-to-English corpus is included to act as a more substantial benchmark as it is a bigger corpus with longer and more complicated sentences than the BTEC Chinese-to-English corpus. A representative example from the ASPEC Japanese-to-English corpus looks as follows:

JA — EBM 堅立のために、精度の高い臨床疫学研究が求められている。

EN — *For the EBM establishment, the clinical epidemiology research of which the accuracy is high has been required.*

### 4.2 Models

In this chapter, we run experiments for and evaluate the following models:

- **Grid Re-encoder** — evaluated in several configurations, without attention
Table 4.1: Comparison of model properties. The Kalchbrenner Re-encoder [Kalchbrenner et al., 2015] and Eriguchi Encoder-Decoder [Eriguchi et al., 2017] are comparable models by other authors who published results on the same corpora used in this thesis.

<table>
<thead>
<tr>
<th>Model</th>
<th>Attention</th>
<th>Re-encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid Re-encoder</td>
<td>Implicit</td>
<td>Yes</td>
</tr>
<tr>
<td>Grid Re-encoder + Att.</td>
<td>Explicit</td>
<td>Yes</td>
</tr>
<tr>
<td>Grid Enc-Dec + Passive att.</td>
<td>Explicit</td>
<td>No</td>
</tr>
<tr>
<td>Grid Enc-Dec + Active att.</td>
<td>Explicit</td>
<td>Yes</td>
</tr>
<tr>
<td>Baseline Enc-Dec</td>
<td>Explicit</td>
<td>No</td>
</tr>
<tr>
<td>Kalchbrenner Re-encoder</td>
<td>Implicit</td>
<td>Yes</td>
</tr>
<tr>
<td>Eriguchi Enc-Dec</td>
<td>Explicit</td>
<td>No</td>
</tr>
</tbody>
</table>

(baseline) and with attention.

- **Grid Encoder-Decoder** – evaluated in several configurations, with passive attention (baseline) and active attention.

- **Baseline Encoder-Decoder** – a model developed by Google [Britz et al., 2017] as part of a general purpose sequence-to-sequence learning framework. It is a standard encoder-decoder with LSTM cells and an attention mechanism. This makes comparisons between the newly proposed models and a standard encoder-decoder network possible. The Baseline Encoder-Decoder comes with built-in beam search, so the effectiveness of beam search can also be evaluated for this model.

Table 4.1 gives quick overview of the attention and re-encoding properties of the various models. Each model has a time complexity of $O(|S||T|)$ where $S$ is the length of the source sentence and $T$ is the length of the target sentence. Note that this estimation of the complexity is an oversimplification. In practice, the computational complexity of the Grid Re-encoder is approximately twice as high as that of the Grid Encoder-Decoder with active attention, because for each target prediction, the Grid Re-encoder reads the source sentence in both forward and backward order, whereas the Grid Encoder-Decoder with active attention only goes over the entire source sentence once. The table also includes two comparable models by other authors who published results on the same corpora, and which will be compared to the models proposed in this thesis. They will be introduced in Section 4.5.2.

### 4.3 Training

All models are trained by minimizing cross-entropy loss using the Adam optimizer [Kingma and Ba, 2014] with a learning rate of 0.001. Mini-batch sizes are chosen
to fit as many sentences in memory as possible. For the BTEC Chinese-to-
English corpus, the Grid Re-encoder uses mini-batches of size 25, which is very
limited due to its high memory needs. For this corpus, the Grid Encoder-Decoder
and the Baseline Encoder-Decoder use mini-batches of size 80. For the ASPEC
Japanese-to-English corpus, the Grid Re-encoder uses mini-batches of size 20 and
the Grid Encoder-Decoder and the Baseline Encoder-Decoder use mini-batches of
size 50. Hidden states and embeddings are initialized using the Xavier initializer
[Glorot and Bengio, 2010], which is based on the number of incoming and outgoing
connections. All models are trained on a single 16-core CPU with 64 GB of RAM
for up to 20 epochs or 10 days.

4.4 Evaluation Metrics

BLEU

BLEU (bilingual evaluation understudy) is a modified n-gram precision used to
measure the quality of a candidate translation given a reference translation [Pap-
ineni et al., 2002]. It is based on precision, which is the proportion of n-grams that
occur in both the candidate and reference translation with respect to the total
number of n-grams in the reference translation. Standard precision is problematic
because any given n-gram may occur many times in the candidate translation and
only occur once in the reference translation, resulting in an inflated score. BLEU
modifies the definition of precision by clipping the counts of each n-gram in the
candidate translation by the number of occurrences in the reference translation.
The BLEU score on an entire (bi-lingual) corpus is computed as follows:

\[ \text{BLEU}_n = \sum_{s \in S} \sum_{n\text{-gram} \in s} \frac{\text{count}_{\text{clip}}(n\text{-gram}, s)}{\sum_{s' \in S'} \sum_{n\text{-gram}' \in s'} \text{count}(n\text{-gram}', s')} \] (4.1)

Here \( S \) is the set of candidate translations and \( S' \) is the set of reference
translations. \( \text{count}_{\text{clip}} = \min(\text{count}(n\text{-gram}, s), \text{count}(n\text{-gram}', s')) \) clips candi-
date counts by the corresponding reference counts. It is important to note that
BLEU is an automated criterion that was designed to compare predictions to one
or more references. However, in the real world, any given sentence can have a
large number of correct translations, and BLEU does not take that into account.
Nevertheless, in order to evaluate a machine translation system it is often most
practical to use an automatic criterion such as BLEU. BLEU was found to cor-
rrelate well with human judgments when using n-grams of up to 4 words long
[Papineni et al., 2002].

Perplexity

Perplexity is based on cross-entropy and can be used to measure how well a
probabilistic model predicts samples. In the case of machine translation, it can
be used to measure how well the translation model can predict words. It indicates a the model’s fluency, and can intuitively be understood as follows: a perplexity per word of $x$ means that on average the model is choosing randomly from $x$ words at any time. A lower value indicates that the model is more certain about its predictions. The perplexity per word for a given sentence $y_1, \ldots, y_T$ is computed as follows, where $K$ is the size of the output vocabulary:

$$PP = 2^{\frac{1}{T} \sum_{t=1}^{T} \sum_{k=1}^{K} y_{t,k} \log(y_{t,k})}$$

(4.2)

4.5 Analysis

We now evaluate the models in several experiments. First, the Grid Re-encoder and Grid Encoder-Decoder are trained in various configurations on the BTEC Chinese-to-English development data in order to find optimal hyperparameters (Section 4.5.1). The most promising parameters are then selected for a more detailed analysis on both the BTEC Chinese-to-English corpus and the ASPEC Japanese-to-English corpus in Section 4.5.2. All experiments are run three-fold and the results are averaged. None of the experiments use any form of ensembling, post-processing of the outputs, or beam search, except the Baseline Encoder-Decoder, which is also run with beam search. For the latter, all reported results are without beam search, unless explicitly stated.

4.5.1 Finding the Optimal Model

The Grid Re-encoder and Grid Encoder-Decoder are both evaluated in various configurations on the BTEC Chinese-to-English corpus. Since a grid search of the complete hyperparameter space is computationally prohibitive, a baseline model is defined for both models based on the results of preliminary experiments, and of which different variations are run to evaluate the effect of different parameter choices.

Experiments are run for state sizes $\in \{300, 450\}$, embedding sizes $\in \{150, 225\}$, and dropout probabilities $\in \{0, 0.25, 0.5\}$. For the Grid Encoder-Decoder, a bigger model with a state size of 600 and embedding size of 300 is also trained, as well as deeper models with 2 layers in both the encoder and the decoder.

Grid Re-encoder

The results of the experiments are summed up in Table 4.2 and Figure 4.2. They show that the baseline model performs well compared to the other experiments in terms of BLEU and perplexity. The bigger model (with a state size of 450) did not manage to converge within 10 days, so the reported result is most likely not the potential final performance, as supported by the trend in the top two plots.
Table 4.2: Grid Re-encoder experiments and results on the BTEC Chinese-to-English development data. If an experiment is marked with an asterisk (*) this means that the experiment did not reach convergence within 10 days.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>300</td>
<td>150</td>
<td>0.5</td>
<td>No</td>
<td>10M</td>
<td>23.8</td>
</tr>
<tr>
<td>Model size*</td>
<td>450</td>
<td>225</td>
<td>0.5</td>
<td>No</td>
<td>18M</td>
<td>22.0</td>
</tr>
<tr>
<td>Dropout</td>
<td>300</td>
<td>150</td>
<td>0.0</td>
<td>No</td>
<td>10M</td>
<td>21.9</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>150</td>
<td>0.25</td>
<td>No</td>
<td>10M</td>
<td>24.3</td>
</tr>
<tr>
<td>Attention*</td>
<td>300</td>
<td>150</td>
<td>0.5</td>
<td>Yes</td>
<td>13M</td>
<td>23.9</td>
</tr>
</tbody>
</table>

Table 4.3: Grid Encoder-Decoder experiments and results (BLEU and perplexity) on the BTEC Chinese-to-English development data.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>300</td>
<td>150</td>
<td>1</td>
<td>0.5</td>
<td>Passive</td>
<td>11M</td>
<td>26.4</td>
</tr>
<tr>
<td>Model size</td>
<td>450</td>
<td>225</td>
<td>1</td>
<td>0.5</td>
<td>Passive</td>
<td>21M</td>
<td>26.6</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>150</td>
<td>2</td>
<td>0.5</td>
<td>Passive</td>
<td>16M</td>
<td>26.4</td>
</tr>
<tr>
<td></td>
<td>450</td>
<td>225</td>
<td>2</td>
<td>0.5</td>
<td>Passive</td>
<td>33M</td>
<td>26.4</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>300</td>
<td>1</td>
<td>0.5</td>
<td>Passive</td>
<td>33M</td>
<td>26.6</td>
</tr>
<tr>
<td>Dropout</td>
<td>300</td>
<td>150</td>
<td>1</td>
<td>0.0</td>
<td>Passive</td>
<td>11M</td>
<td>24.7</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>150</td>
<td>1</td>
<td>0.25</td>
<td>Passive</td>
<td>11M</td>
<td>26.0</td>
</tr>
<tr>
<td>Attention</td>
<td>300</td>
<td>150</td>
<td>0.5</td>
<td>Active</td>
<td>12M</td>
<td>26.9</td>
<td>5.87</td>
</tr>
<tr>
<td></td>
<td>450</td>
<td>225</td>
<td>1</td>
<td>0.5</td>
<td>Active</td>
<td>23M</td>
<td>26.6</td>
</tr>
</tbody>
</table>

in Figure 4.2. We do observe that the bigger model converges faster than the baseline.

The default dropout probability of 0.5 appears to be slightly high; a value of 0.25 leads to slightly better performance. Since in this case the two are so close, it can reasonably be expected that the higher value will potentially generalize better. No dropout leads to faster convergence at the cost of generalization performance on the development data, indicating an overfit.

The attention mechanism speeds up training, but its final BLEU score is similar to that of the baseline model. This suggests that the attention mechanism may potentially be redundant. However, the BTEC Chinese-to-English corpus mostly consists of short sentences, so attention is likely not very important, making it unclear whether or not the attention mechanism can be beneficial. Using the attention mechanism does lead to a higher perplexity; indicating that the model is less certain about its predictions than without it.

**Grid Encoder-Decoder**

The results of the experiments are summed up in Table 4.3 and Figure 4.3. Here too, the baseline model mostly performs on par with the other experiments. Changing the size of the model does not seem to have a big impact on performance. Figure 4.3 shows that increasing the size of the state and embeddings
4.5. Analysis

Model size

![Graph showing BLEU scores for different model sizes.](image)

Dropout

![Graph showing cross-entropy loss for different dropout probabilities.](image)

Attention

![Graph showing BLEU scores for different attention mechanisms.](image)

Figure 4.2: BLEU scores (left) and cross-entropy loss (right) the Grid Re-encoder on the BTEC Chinese-to-English development data. * marks the baseline model.

speeds up training, whereas adding more layers to the network does not seem to have a noticeable effect.

The default dropout probability of 0.5 appears to be superior to lower dropout probabilities. Performance on the development data converges prematurely with a dropout probability of 0.25, and no dropout leads to a significant overfit on the train data where performance on the development data decreases while the training loss keeps decreasing.
Chapter 4. Experiments

The active attention mechanism performs slightly better than the passive attention mechanism in terms of BLEU, but in terms of perplexity it is slightly worse. Figure 4.3 shows that active attention speeds up training, but converges to a similar BLEU score. How do we know that the active attention mechanism does not simply improve performance because it adds more parameters to the model? From the model size experiments we conjecture that this is likely not the case, since simply increasing state sizes or adding layers seems to have a very small effect on performance, whereas active attention has a more noticeable effect. The effectiveness of active attention will become more clear on a more mature corpus, as we will see in the next section.

4.5.2 Translation Performance

The best parameter choices are picked from the results in Section 4.5.1 for both models and combined to train optimal models on both the BTEC Chinese-to-English corpus and the ASPEC Japanese-to-English corpus. The effect of attention in the Grid Re-encoder is tested by training one model with attention and one without. In the case of the Grid Encoder-Decoder one model is trained with active attention and one with passive attention. The models are then compared to the Baseline Encoder-Decoder. The following configurations are used:

- **Grid Re-encoder**: state size = 300, embedding size = 150, dropout probability = 0.5, without/with attention.
- **Grid Encoder-Decoder (BTEC)**: state size = 450, embedding size = 225, dropout probability = 0.5, passive/active attention.
- **Grid Encoder-Decoder (ASPEC)**: state size = 300, embedding size = 150, dropout probability = 0.5, passive/active attention.
- **Baseline Encoder-Decoder**: state size = 450, embedding size = 225, dropout probability = 0.5, beam size \( \in \{1, 15\} \).

For completeness, the results of several other authors are included where possible. For the BTEC Chinese-to-English corpus, this is the result of the Grid LSTM translation model by [Kalchbrenner et al., 2015] (referred to as *Kalchbrenner Re-encoder*). The size of the hidden state and memory vector of this model is 450, the size of the word embeddings is unknown. Kalchbrenner Re-encoder uses beam search decoding with a beam of size 20 and an ensemble of 7 models, of which neither are used in the Grid Re-encoder experiments of this thesis. This means that the results of [Kalchbrenner et al., 2015] are not directly comparable to the results reported here. The total number of parameters of the model is unknown, but likely similar to that of the Grid Re-encoder. For the ASPEC Japanese-to-English corpus, the results are compared to that of the attention-based encoder-decoder model by [Eriguchi et al., 2017], referred to as *Eriguchi*
4.5. Analysis

Model size

![Model size chart]

Dropout

![Dropout chart]

Attention

![Attention chart]

Figure 4.3: BLEU scores (left) and cross-entropy loss (right) the Grid Encoder-Decoder on the BTEC Chinese-to-English development data. * marks the baseline model.

Encoder-Decoder. In this model, each network has one LSTM layer with a state size of 256 and the word embeddings are also of size 256. Here too the total number of parameters is unknown, but likely comparable to that of the baseline Grid Encoder-Decoder. The Ergiguchi Encoder-Decoder also uses beam search, which should be taken into account when comparing results.
Table 4.4: 4-gram (which also includes 1,2,3-grams) and 1-gram BLEU scores and perplexities on the BTEC Chinese-to-English test data.

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU-4</th>
<th>BLEU-1</th>
<th>Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid Re-encoder</td>
<td>32.1</td>
<td>62.3</td>
<td>4.90</td>
</tr>
<tr>
<td>Grid Re-encoder + Att.</td>
<td>32.4</td>
<td>64.6</td>
<td>6.55</td>
</tr>
<tr>
<td>Kalchbrenner Re-enc. + Beam + Ensemble</td>
<td>42.4</td>
<td>64.0</td>
<td>4.54</td>
</tr>
<tr>
<td>Grid Enc-Dec + Passive att.</td>
<td>34.6</td>
<td>64.0</td>
<td>4.68</td>
</tr>
<tr>
<td>Grid Enc-Dec + Active att.</td>
<td>34.9</td>
<td>64.5</td>
<td>5.84</td>
</tr>
<tr>
<td>Baseline Enc-Dec</td>
<td>34.3</td>
<td>65.3</td>
<td></td>
</tr>
<tr>
<td>Baseline Enc-Dec + Beam (15)</td>
<td>35.7</td>
<td>69.5</td>
<td></td>
</tr>
</tbody>
</table>

BTEC Chinese-to-English

Table 4.4 gives the performance of all models on the BTEC Chinese-to-English corpus. In general, the models proposed in this thesis underperform compared to the result of the Kalchbrenner Re-encoder, but that model uses both ensembling and beam search, contrary to the Grid Re-encoder and Grid Encoder-Decoder.

As expected based on previous experiments, attention slightly improves the Grid Re-encoder’s performance in terms of BLEU, at the cost of a higher perplexity. From this result, it remains unclear whether or not attention improves over the basic version, in which attention is implicit due to re-encoding.

The Grid Encoder-Decoder outperforms the Grid Re-encoder by roughly 1.5 BLEU points and also reaches a lower perplexity in both configurations, in both BLEU and perplexity. The active attention mechanism slightly improves performance in terms of BLEU, but not in terms of perplexity.

The Baseline Encoder-Decoder reaches a performance that is on par with the Grid Encoder-Decoder with passive attention, which is not surprising considering that their architectures are quite similar. Beam search adds 1.4 BLEU points compared to greedy decoding, so it provides a substantial improvement.

As stated before, the results on the BTEC Chinese-to-English corpus should be interpreted with caution. It is the expectation that attention and re-encoding will show their true potential when used on longer sentences, as supported by the results on the ASPEC Japanese-to-English corpus, and this will be shown in the next subsection.

ASPEC Japanese-to-English

The ASPEC Japanese-to-English corpus reveals bigger differences between various models and configurations, as shown in Table 4.5. The Grid Re-encoder did not manage to converge within 10 days, but showed potential for better performance. Compared to the Eriguchi Encoder-Decoder, the models proposed in this thesis still underperform, but they come close.
Table 4.5: 4-gram (which also includes 1,2,3-grams) and 1-gram BLEU scores and perplexities on the ASPEC Japanese-to-English test data. Models marked with an asterisk (*) did not fully converge within 10 days, so their results are not definitive.

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU-4</th>
<th>BLEU-1</th>
<th>Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid Re-encoder*</td>
<td>16.9</td>
<td>52.0</td>
<td>11.84</td>
</tr>
<tr>
<td>Grid Re-encoder + Att.*</td>
<td>12.0</td>
<td>45.5</td>
<td>73.26</td>
</tr>
<tr>
<td>Grid Enc-Dec + Passive att.</td>
<td>13.0</td>
<td>46.7</td>
<td>14.41</td>
</tr>
<tr>
<td>Grid Enc-Dec + Active att.</td>
<td>15.7</td>
<td>51.1</td>
<td>14.17</td>
</tr>
<tr>
<td>Baseline Enc-Dec</td>
<td>17.0</td>
<td>50.3</td>
<td></td>
</tr>
<tr>
<td>Baseline Enc-Dec + Beam (15)</td>
<td>18.3</td>
<td>53.7</td>
<td></td>
</tr>
<tr>
<td>Eriguchi Enc-Dec + Beam</td>
<td>17.9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notably, the Grid Re-encoder reaches a BLEU score of 16.9, which is close to the score of the Eriguchi Encoder-Decoder, and significantly better than the score of the Grid Encoder-Decoder. Given more training, the result would likely have improved, although its current result is still striking. Another notable result is that it performs much better without the attention mechanism. With it, the model seems unable to learn properly, as shown by the low BLEU score and extremely high perplexity. The latter suggests that the attention mechanism only serves to confuse the model. It looks like explicit attention as proposed in this thesis may not be necessary for the Grid Re-encoder.

For the Grid Encoder-Decoder, active attention leads to significantly better performance in terms of BLEU compared to passive attention, supporting (but not proving) the claim that active attention, and re-encoding in general, can be beneficial especially with longer sentences. A more in-depth investigation into sentence lengths is carried out in the next section and gives more supportive evidence.

The Baseline Encoder-Decoder outperforms all of the other models, including the Eriguchi Encoder-Decoder, with a BLEU score of 18.3 with beam search. Without beam search, the score goes down to 17.0. Simply put, the Baseline Encoder-Decoder is a very strong baseline. Again, the Grid Re-encoder’s final performance is unknown, so it is unclear which model is better. The difference in performance with the Grid Encoder-Decoder may possibly explained by differences in implementation and the choice of (Grid) LSTM unit. Perhaps the simplicity of the standard LSTM is more useful than Grid LSTM in the end.

**Sentence Length vs. Performance**

To investigate whether or not the attention mechanism improves over the basic version in the case of the Grid Re-encoder, and active attention over passive attention in the case of the Grid Re-encoder, a BLEU score is computed for various
ranges of source sentence lengths on both the BTEC Chinese-to-English corpus and the ASPEC Japanese-to-English corpus. Figure 4.4 shows the results. The left plot shows that across different source sentence lengths all model variations perform roughly the same. It shows a decreasing trend line: as the source sentence length increases, the BLEU score decreases too. There is no considerable difference between the Baseline Encoder-Decoder and the other models. It is possible that the models cannot handle longer sentences properly due to their design, but it may also be explained by the lack of longer sentences in the train data, as shown in Figure 4.1 on the left, which in itself can cause the model to perform worse on longer sentences.

This claim is supported by the plot on the right in Figure 4.4, which shows that the performance across sentence lengths on the ASPEC Japanese-to-English corpus stays much more constant. For the ASPEC Japanese-to-English corpus, the best performing model on long sentences is the Grid Re-encoder without attention. The Grid Re-encoder with attention performs slightly worse, suggesting that the model probably does not need an explicit attention mechanism; its implicit attention that is inherent to deep re-encoding could be sufficient. For the Grid Encoder-Decoder active attention yields a substantial improvement over passive attention on longer sentences. The Baseline Encoder-Decoder, which showed the strongest final performance over all sentence lengths, is in fact one of the worst performing models on longer sentences. It seems that its strong performance mainly comes from short sentences, where it is the superior model. All things considered, it seems likely that re-encoding can indeed improve translation quality of longer sentences.
4.5. Analysis

Inspection of BTEC Chinese-to-English Translations

Inspection of generated predictions by the models reveals that a the models make a number of common mistakes. Many neural models suffer from the common problem that is over-translation: they unnecessarily produce output words. In many cases, the model repeats the same fragment multiple times, indicating that it may be unable to keep track of translation coverage properly.

In general, the Grid Encoder-Decoder with active attention seems to convey the meaning of the sentence better than the Grid Encoder-Decoder with passive attention and the Baseline Encoder-Decoder, presumably because the active attention mechanism allows it to discriminate between source words better (see e.g., sentences 3 and 4). The Grid Re-encoder with attention generally seems to produce less coherent translations than the Grid Re-encoder without attention.

1. Grid Re-encoder
   how about lunch with me for lunch ? i lost a good restaurant near my hotel .

   Grid Re-encoder + Attention
   what time is lunch with lunch lunch ? i found a nice restaurant near my hotel near my hotel .

   Grid Encoder-Decoder + Passive attention
   how about lunch ? i suggest that i went to a restaurant near the hotel near the hotel .

   Grid Encoder-Decoder + Active attention
   how about having lunch together ? i saw a good restaurant near my hotel near the hotel .

   Baseline Encoder-Decoder
   how about going to dinner ? i found a good restaurant around the hotel .

   Reference
   how about having lunch with me some day ? i found a good restaurant near my hotel .

2. Grid Re-encoder
   i know what you feel like i hear that he said are in trouble .

   Grid Re-encoder + Attention
   i know your feeling , i hear he was playing school and have trouble .

   Grid Encoder-Decoder + Passive attention
   i understand that i ’ve heard that i ’ve heard in mary and i have been in the school .

   Grid Encoder-Decoder + Active attention
   i understand your understanding . i heard him in mary .
Baseline Encoder-Decoder
i understand you know what i understand him i have a problem in school.

Reference
i know how you feel. i heard he got in trouble at school again.

3. Grid Re-encoder
the japan is a very orange wooden turbulence, and it will be completely behind the _UNK_.

Grid Re-encoder + Attention
japan declaration rooms are a very close ito and anyone who would be completely up-to-date down.

Grid Encoder-Decoder + Passive attention
the bath is a bath with a bath with a _UNK_, but it was closed.

Grid Encoder-Decoder + Active attention
the bathroom in japan is a hot machine with everybody's completely in it.

Baseline Encoder-Decoder
the bath is a UNK stage, however, the UNK has been in UNK.

Reference
the japanese bath is a deep bathtub where one can really soak.

4. Grid Re-encoder
would you like to have to have dinner on french fries?

Grid Re-encoder + Attention
would you like to eat hamburger into french fries or others?

Grid Encoder-Decoder + Passive attention
would you like to have a _UNK_ or french fries or french fries?

Grid Encoder-Decoder + Active attention
would you like to have a hamburger with french fries or some others?

Baseline Encoder-Decoder
would you like to have a hamburger or french fries or french fries?

Reference
would you like french fries or something with your burger?
4.5. Analysis

5. Grid Re-encoder
   i’m afraid i did n’t catch your best.

Grid Re-encoder + Attention
   i’m afraid i did n’t say you say.

Grid Encoder-Decoder + Passive attention
   i’m afraid i did n’t tell you something about you.

Grid Encoder-Decoder + Active attention
   i’m afraid i did n’t follow what you said.

Baseline Encoder-Decoder
   i’m afraid i do n’t have to explain you.

Reference
   i’m afraid i do n’t understand what you’re saying.

Inspection of ASPEC Japanese-to-English Translations

The models make the same type of mistakes on the ASPEC Japanese-to-English corpus. Over-translation and repetition are recurring problems. The high number of repetitions suggests that the models may still have problems remembering which source words it has attended to already (and which predictions it has made), indicating that the proposed models still need further refinements.

1. Grid Re-encoder
   The dose rate of “_UNK_ power” can be calculated for details using _UNK_ software.

Grid Re-encoder + Attention
   Dose rate “GE power generating stations can be calculated in detail using _UNK_ software.

Grid Encoder-Decoder + Passive attention
   Using the _UNK_ software, the dose rate of “Fugen” can be calculated in detail.

Grid Encoder-Decoder + Active attention
   Using the software software, the dose rate of “_UNK_” is possible in detail.

Baseline Encoder-Decoder
Chapter 4. Experiments

The dose rate of “Fugen generator” can be calculated in detail using UNK software.

Reference
Details of dose rate of “Fugen Power Plant” can be calculated by using DERS software.

2. Grid Re-encoder
   This paper describes production and characteristic evaluation of this detector.

   Grid Re-encoder + Attention
   Production and characteristic evaluation of this detector are described.

   Grid Encoder-Decoder + Passive attention
   This paper describes the production and characteristics of this detector.

   Grid Encoder-Decoder + Active attention
   The production and characteristics evaluation of this detector are described.

   Baseline Encoder-Decoder
   The manufacture and characteristic evaluation of this detector are described.

   Reference
   The fabrication and property evaluation of this detector were described.

3. Grid Re-encoder
   The interaction between molecules and molecules affecting the ultrastructure structure greatly influences.

   Grid Re-encoder + Attention
   Interaction between morphology and molecules of molecules greatly influences in self tissue structure.

   Grid Encoder-Decoder + Passive attention
   The molecular interaction between molecule and molecular interaction greatly influence the self-organization of the self-organization.

   Grid Encoder-Decoder + Active attention
   The intermolecular interaction between molecules and molecules greatly affects the self structure.

   Baseline Encoder-Decoder
   The molecular interactions between molecular and molecules greatly influences the molecular structure.

   Reference
4.5. Analysis

Molecular shape and intermolecular interaction influence self-assembled structures greatly.

4. Grid Re-encoder

When the nitrogen discharge light is irradiated, large Bragg pulses penetrate discharge light, _UNK_ pulse was proven from the _UNK_ figure of discharge current.

**Grid Re-encoder + Attention**

When large intermittent pulses illuminated by nitrogen discharge light irradiation discharge light, _UNK_ pulse proven from _UNK_ figures of discharge current.

**Grid Encoder-Decoder + Passive attention**

The pulsed anode pulse was irradiated when irradiating the nitrogen discharge light, when the discharge of the discharge is irradiated, and it was proven from the _UNK_ of the discharge current.

**Grid Encoder-Decoder + Active attention**

When irradiating a pulsed discharge of nitrogen discharge, discharge pulse pulse duration was found to be _UNK_ by the _UNK_ of the discharge current.

**Baseline Encoder-Decoder**

When a large intermittent pulse is irradiated when nitrogen discharge light is irradiated, it was proven that the UNK pulse can be irradiated by the UNK of the discharge current, when the discharge light is irradiated.

**Reference**

When large intermittent pulses radiate discharge light, and when nitrogen discharge light is not radiated, it was proven that it changed to small group pulses from Lissajou’s figure of discharge current.

5. Grid Re-encoder

From the data, the segment preparation of the eyeball tissue was performed, and mesh preparation was performed by the numerical application of individual data.

**Grid Re-encoder + Attention**

From the data, segments of eyeball tissue was carried out, mesh preparation was carried out by numerical commercialization of individual data.

**Grid Encoder-Decoder + Passive attention**

From the data, the segmentation of the eyeball tissue was carried out, and the mesh preparation was carried out by the digitization of individual data.

**Grid Encoder-Decoder + Active attention**

The mesh segment was carried out from the data, and mesh mesh was carried out by the digitization of individual data.
Baseline Encoder-Decoder

From the data, the segment organization was carried out, and the mesh preparation was carried out by the numerical of individual data.

Reference

The eyeball tissue was segmented from the data, and the mesh was prepared by digitization of individual data.
Chapter 5

Conclusion

This thesis proposed and investigated two models for neural machine translation that make use of re-encoding and attention. The Grid Re-encoder uses re-encoding and is extended with an attention mechanism. The Grid Encoder-Decoder is extended with an active attention mechanism that re-encodes and attends to source words. Both models were evaluated in various experiments on two corpora and compared to the Baseline Encoder-Decoder model with attention. In general, the results showed that re-encoding can be beneficial, especially as sentences get longer. However, the models are certainly not perfect yet, as some of the problems re-encoding aims to solve are still present. Specifically, repetition and over-translation remain a (somewhat reduced) problem.

The Grid Re-encoder was shown to perform relatively well. On the BTEC Chinese-to-English corpus, its result did not stand out and did not provide us with useful insights, which was possibly caused by the imbalance of the corpus, as well as the lack of longer sentences. However, it showed a surprising result on the ASPEC Japanese-to-English corpus, where it outperformed the Grid Encoder-Decoder by a large margin. This result suggests that the model is able to show its true potential on longer sentences, rather than the short sentences that the BTEC Chinese-to-English corpus mainly consists of.

Neural attention was designed to improve translation quality on longer sentences. The addition of an explicit attention mechanism to the Grid Re-encoder turned out to be unnecessary. The basic version of the model is able to implicitly attend to source words by repeatedly re-encoding the source sentence, and the attention mechanism as proposed in this thesis did not provide an improvement. In fact, on the ASPEC Japanese-to-English it confused the model and worsened its performance.

The model’s biggest flaw is related to its computational complexity. The depth of this model is $T \times S \times 2$, where $T$ and $S$ are the lengths of the target and source sentence respectively, and the source sentence is processed twice (in both directions). This means that training the Grid Re-encoder on the relatively small
BTEC Chinese-to-English corpus takes many days, and that the model scales badly to more mature corpora. While re-encoding in itself can be beneficial, perhaps it is not necessary to have such a deep model for machine translation [Kalchbrenner et al., 2016].

The Grid Encoder-Decoder is computationally less complex and showed promising results. The model with passive attention suffered from longer sentences, similarly to the Baseline Encoder-Decoder. Qualitative analysis of the results showed that this model tends to over-translate and repeat itself. This may potentially and partly be explained by a lack of variance in traditional encodings, which restricts the model in discriminating between source words, and the limited ability to remember which words have been translated. The addition of the active attention mechanism seems to reduce these problems and allows the model to produce higher quality translations, especially on longer sentences.

**Future Work**

Re-encoding has not fully solved the problems of under-translation, over-translation and repetition. The idea behind re-encoding is that it allows the model to remember additional information about the source and target sentences more easily, such as translation coverage and predictions made so far. Remembering which predictions have been made can also be captured in a different way. Likely, this information is currently stored in the state of the decoder, but as sentences get longer this information may fade. Therefore, it could be useful to have an additional mechanism that provides the model with attention over previous decoder states. In this way, the model would first attend to its decoder states (previous predictions), re-encode the source sentence accordingly, and then decide which word(s) to translate.

All in all, it seems clear that attention is an essential ingredient for neural machine translation [Bahdanau et al., 2015, Luong et al., 2015, Vaswani et al., 2017]. With attention, the model generates a unique source sentence representation for each target prediction by taking a weighted sum of the source encodings. However, the standard attention mechanism attempts to focus on a single source word by design, due to the use of a softmax. Sometimes, this is problematic, as multiple source words may be relevant depending on the next prediction. Following [Kaiser and Bengio, 2016], it would be interesting to see if it is possible to come up with a mechanism that re-encodes and attends to the entire source sentence in parallel.

Another issue that deserves attention is computational efficiency. While the computational complexity of the Grid Encoder-Decoder is an improvement over the computational complexity of the Grid Re-encoder, it remains problematic. It would be interesting to see how well the Grid Encoder-Decoder would perform with the standard, less complex LSTM units (and some small, necessary modifications) instead of Grid LSTM units. Alternatively, it would be interest-
ing to look at convolutional neural networks. Whereas recurrent neural networks are constrained by temporal relationships, convolutional neural networks can act on entire sentences in parallel, which can greatly speed up computations. They have been shown to be applicable in neural machine translation [Gehring et al., 2016, Kalchbrenner et al., 2016, Kaiser and Bengio, 2016]. Potential future improvements of the Grid Encoder-Decoder would include replacing the computationally expensive Grid LSTMs of the encoder and re-encoder with convolutional neural networks.


