Master Thesis

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Incorporating Language Models into Non-autoregressive Neural Machine Translation

Bc. Zdeněk Kasner

Supervisor: Mgr. Jindřich Helcl
Field of study: Open Informatics
Subfield: Artificial Intelligence
May 2019

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kasner** Jméno: **Zdeněk** Osobní číslo: **420815**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Použití jazykových modelů v neautoregresivním neuronovém strojovém překladu

Název diplomové práce anglicky:

Incorporating Language Models into Non-autoregressive Neural Machine Translation.

Pokyny pro vypracování:

Cílem práce je prozkoumat metody neautoregresivního neuronového strojového překladu. Výhodou neautoregresivních metod je vysoká rychlost dekódování, které se dosáhne na úkor kvality překladu. Práce se zaměří na možnosti kombinování neautoregresivních modelů s jazykovými modely, které by mohly pomoci kvalitu překladu zvýšit při zachování časové náročnosti. Projekt bude vycházet z článku "End-to-End Non-Autoregressive Neural Machine Translation with Connectionist Temporal Classification" a bude implementován pomocí nástroje Neural Monkey.

Seznam doporučené literatury:

Libovický, J., & Helcl, J. (2018). End-to-End Non-Autoregressive Neural Machine Translation with Connectionist Temporal Classification. arXiv preprint arXiv:1811.04719.

Gu, J., Bradbury, J., Xiong, C., Li, V. O., & Socher, R. (2017). Non-autoregressive neural machine translation. arXiv preprint arXiv:1711.02281.

Lee, J., Mansimov, E., & Cho, K. (2018). Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement. arXiv preprint arXiv:1802.06901.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in Neural Information Processing Systems(pp. 5998-6008).

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006, June). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In Proceedings of the 23rd international conference on Machine learning (pp. 369-376). ACM.

Helcl, J., & Libovický, J. (2017). Neural Monkey: An open-source tool for sequence learning. The Prague Bulletin of Mathematical Linguistics, 107(1), 5-17.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Jindřich Helcl, katedra počítačů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **28.01.2019** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **20.09.2020**

Jindřich Helcl podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Acknowledgements

I would like to thank my supervisor Mgr. Jindřich Helcl who provided me with all the guidance and expertise necessary for the thesis and helped me whenever I needed. The same applies for Mgr. Jindřich Libovický and other people from ÚFAL who participated in the projects I built upon.

My gratitude also goes to my friends and family for the continuous support, welcoming environment, and proofreading of this thesis.

Last but not least, I would like to thank the people from the AI center of the Faculty of Electrical Engineering for all the knowledge I gained during my master's studies and the people at KU Leuven who brought me to the field of natural language processing.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

Prague, 10. May 2019

Abstract

In order to improve the fluency of a non-autoregressive model for neural machine translation, we propose an extension for the scoring model used during the beam search decoding. We compute the score as a linear combination of feature values, including the score from an n-gram language model and other auxiliary features. We determine the weights of the features using the structured perceptron algorithm. We train the models for three language pairs and evaluate their decoding speed and translation quality. The results show that our proposed models are still efficient in terms of decoding speed while achieving a competitive score relative to autoregressive models.

Keywords: neural machine translation, non-autoregressive machine translation, beam search, language model, structured perceptron

Abstrakt

V této práci navrhujeme způsob pro zlepšení plynulosti výstupu neautoregresivního modelu pro neuronový strojový překlad. Využíváme k tomu rozšířený model pro počítání skóre během paprskového prohledávání. Skóre vypočítáváme jako lineární kombinaci dílčích skóre pocházejících z n-gramového jazykového modelu a dalších pomocných příznaků. Váhy pro lineární kombinaci určujeme pomocí strukturovaného perceptronu. Pro vyhodnocení rychlosti a kvality překladu trénujeme modely pro tři dvojice jazyků. Výsledky ukazují, že modely s navrženým vylepšením jsou stále dostatečně efektivní z hlediska rychlosti a zároveň dosahují výsledků srovnatelných s autoregresivními modely.

Klíčová slova: neuronový strojový překlad, neautoregresivní strojový překlad, paprskové prohledávání, jazykový model, strukturovaný perceptron

Contents

Figures

Tables

Chapter 1

Introduction

Advances in artificial intelligence (AI) allow machine translation (MT) to substitute the work of human translators in everyday life. In 2018, more than 143 billion words per day were translated by Google Translate [1]. Demand for fast and high-quality MT makes it one of the current research targets.

Recently, end-to-end neural machine translation (NMT) models based on deep neural networks (DNNs) outperformed traditional statistical machine translation (SMT) approaches in translation quality [2, 3]. At the same time, evaluating DNNs is computationally expensive which makes the NMT models slower than SMT models during the inference time. Computations in neural networks can be parallelized, which improves the inference speed in practice. However, some parts of the current NMT models cannot be parallelized. This is caused by the *autoregressive* nature of decoders used in the models. Autoregressive decoders generate output tokens sequentially, conditioning the output in each timestep on the previously generated sequence of tokens. This implies that the decoding is not parallelizable and the decoding time is linear in the length of the output sequence.

Non-autoregressive MT tackles the low decoding speed by removing the autoregressive property of a decoder. In non-autoregressive models, each output token is independent on the rest of the output sequence. While it allows parallelization of the decoding process, it results in lower quality of the translation as the decoder cannot capture the sequential nature of natural language.

The goal of this thesis is to improve the translation quality of non-autoregressive models, making it comparable with autoregressive models while preserving their main advantage – the decoding speed. To achieve the goal, we improve the decoding process in a non-autoregressive model with Connectionist Temporal Classification (CTC) described in [4]. Instead of decoding output tokens using only the score from the model, we compute the score as a linear combination of feature values. We include the score from the model as one of the features and we design the additional features to improve the quality of the output sentence: a language model helps to improve the

fluency of the sentence, the rest of the features help to generate the sentence of correct length. To learn the feature weights, we use a variant of structured perceptron algorithm for beam search [5].

We evaluate the benefits of suggested improvements experimentally. We train the enhanced MT models for three language pairs, evaluate the models on standard datasets, and compare the results with state-of-the-art baselines.

The results show that our approach can be used not only to increase the quality of non-autoregressive NMT, but also to control the tradeoff between NMT speed and quality.

The rest of the thesis is organized as follows:

- **Chapter 2** introduces the field of MT, summarizes briefly its history and provides an overview of MT algorithms and evaluation metrics.
- **Chapter 3** compares current approaches to non-autoregressive MT and introduces the model used as a base model in the thesis.
- **Chapter 4** presents the improvements over the base model and explains the design choices for the features, the feature weights, and the scoring model.
- **Chapter 5** describes the framework used for implementing and evaluating the experiments, and other technologies and libraries used in the process.
- **Chapter 6** provides an overview of the experiments and discusses the results and the benefits of the suggested approach.
- **Chapter 7** concludes the thesis, summarizes the work, and suggests directions for future work.

Chapter 2

Machine Translation

This chapter provides a brief overview of the history of MT, followed by the introduction of two MT paradigms – SMT and NMT. Concepts of MT which are essential for understanding the main contribution of this thesis – as a language model, encoder-decoder approach, the Transformer model with self-attentive layers, or the beam search algorithm – are explained here. The chapter also introduces evaluation metrics for MT.

2.1 Background

MT is an application of computers for automated translation of text from one language into another [6]. The principal reason for automating the translation process is the ubiquitous need for translation in the today's interconnected world and the lack of human translators to perform it. Advantages of having automated translation may include better international cooperation, removal of language barriers, and military or economy motives. There is also pure research motivation – MT allows us to study how the human brain works and how to formalize and automate working with language. According to [7], MT is one of the *AI-complete* tasks, i.e. solving the problem completely involves understanding the basics of human intelligence, our common sense, and knowledge acquisition.

History of MT (described in [8, 9]) follows closely the history of AI. First attempts to use a computer for automatic translation dates back to the invention of the first computers in 1940's. At that time, MT was purely rule-based – machines were equipped with hand-coded rules for manipulating with words and sentences. Creating rule-based algorithms required considerable amount of human expertise and the algorithms were not robust. Due to the limited computing power and lack of available datasets, this was the only available approach until 1980's.

In 1990's, the statistical approach to MT revived the interest in MT.

Large corpora[1] were used for automated capturing of patterns in language. SMT allowed limiting the amount of human involvement and knowledge needed for MT. Still, it required many simplifications to make the translation computationally tractable and multiple processing steps were needed to achieve the state-of-the-art results. SMT is discussed in Section 2.2.

DNNs have changed the field of MT and natural language processing (NLP) in general. After the introduction of the first end-to-end models based on DNNs as [10] in 2013 and [11, 12] in 2014, NMT has achieved and surpassed the state-of-the-art results and replaced previous MT models in commercial translation systems. NMT is characterized by the end-to-end approach, requiring minimal or no preprocessing. This effectively involves training a single, large neural network that reads an input sentence and outputs its correct translation [2]. NMT is discussed in Section 2.3.

2.2 Statistical Machine Translation

SMT systems are designed to learn automatically from data. The translation task in SMT is defined as follows: given a sentence f in the target language F, find the most probable sentence e in the source language E, i.e. the sentence which maximizes $p(e|f)$. The translation in SMT is viewed as a deterministic decoding process – each sentence f is an encoded version of the sentence e and the job of the translation system is to decode it [13]. Although this approach involves simplifications ignoring the semantics and pragmatics of the translation process, it has proven successful in practice.

A standard way of determining e with the highest conditional probability $p(e|f)$ builds on the Bayes' theorem [14, 15].

$$e = \arg\max_e p(e|f) = \arg\max_e \frac{p(f|e)p(e)}{p(f)} = \arg\max_e p(f|e)p(e). \tag{2.1}$$

Factorizing the conditional probability allows to split the translation process. First, we train two independent models – a *source language model* for estimating $p(e)$ and a *translation model* for estimating $p(f|e)$. The parameters of both models are estimated automatically from corpora, but possibly in a different way. Then we use a *decoder* to perform the actual translation, i.e. to find the sentence which maximizes the product $p(e)p(f|e)$.

2.2.1 Language Model

A language model (LM) is used in SMT to determine $p(e)$, i.e. the probability of the sentence e in the source language E. It should assign high probabilities

[1] *corpus*, pl. *corpora* – a dataset of natural language examples

to the sentences which are fluent in the source language and low probabilities to any other sentences.

To compute $p(e)$, we split the sentence e into the sequence of tokens: $e = (e_1, e_2, \ldots, e_m)$. Using this notation, the probability of the sentence can be written down as a product of conditional probabilities for each token:

$$p(e) = p(e_1) \cdot p(e_2|e_1) \cdot p(e_m|e_1, e_2, \ldots, e_{m-1}) = \prod_{i=1}^{m} p(e_i|e_1, \ldots, e_{m-1}). \tag{2.2}$$

Estimating all conditional probabilities would require a large number of parameters. To make the process computationally tractable, we can make a simplifying assumption and limit the history of each token. This gives rise to *n-gram LMs*, where the probability of each token is conditioned only on its $n-1$ immediate predecessors; n is the *order* of the LM.

The probability of each n-gram can be calculated from the data by counting its occurrences in the corpus and normalizing the count. For a bigram model ($n = 2$):

$$p(e_2|e_1) = \frac{\#(e_1 e_2)}{\#(e_1)}. \tag{2.3}$$

Some n-grams may be missing in the corpus. There are several ways to deal with this problem, e.g. using a linear interpolation with lower-order n-grams:

$$p(e_2|e_1) = \lambda \frac{\#(e_2)}{N} + (1-\lambda)\frac{\#(e_1 e_2)}{\#(e_1)}, \tag{2.4}$$

where N is the total number of words in the corpus and $\lambda \in (0, 1)$.

The probability of the sentence $p(e)$ is calculated as a product of probabilities of all the n-grams in the sentence:

$$p(e) = \prod_{i=1}^{m} p(e_i|e_{m-n+1}, \ldots, e_{m-1}). \tag{2.5}$$

Special tokens `<s>` are used as a padding in the beginning of the sentence.

2.2.2 Translation Model

A translation model in SMT is used to estimate $p(f|e)$, i.e. the probability of the sentence f being a translation of the sentence e. It assumes that the words in the source language E generate the words in the target language F. In the example pair of sentences *(Jean aime Marie | John loves Mary)* *John* generates *Jean*, *loves* generates *aime*, and *Mary* generates *Marie*. This correspondence between the words is called an *alignment*.

Intuitively, the alignment has to be neither in order, nor one-to-one. Therefore, the translation model needs to capture several parameters [16]:

- **lexical probability** t – probability that a word from E translates to a word from F
- **fertility** n – probability that a word from E generates a certain number of words in F
- **distortion** d – probability that a word from F on the position i corresponds to a word from E on the position j, given the lengths of e and f.

For example, $t(Jean|John)$ is the probability that *John* translates into *Jean*, $n(2|John)$ is the probability that *John* generates 2 words and $d(2|1,3,3)$ is the probability that the word on the position 2 in f is generated by the word on the position 1 in e given that $|e| = 3$ and $|f| = 3$. These parameters can be estimated iteratively from the data using the expectation-maximization algorithm [17].

2.2.3 Decoder

Decoder in SMT aims to find a sentence e maximizing $p(e)p(f|e)$. There are too many possible sentences to perform an exhaustive search. However, we can get good results in practice by building a solution incrementally, keeping only a limited number of solutions with the highest probability in each step. This approach is a variant of the *beam search* algorithm described in detail in Section 2.5.

2.3 Neural Machine Translation

NMT is a data-driven approach to MT which adopts *neural networks* to train the statistical models. Neural networks are a powerful machine learning technique used to achieve state-of-the-art results on difficult problems such as visual object recognition [18] and speech recognition [19].

Similarly to SMT, the decoding phase in NMT is based on looking for the translation with the highest probability. The advantage of the NMT over SMT models is the ability of neural networks to learn complex functions, which eliminates many of the simplifying assumptions in SMT models. NMT models are able to condition the target translation on the whole source sentence and capture non-trivial long-distance dependencies. Moreover, general focus in NMT is to train the models *end-to-end* which removes the implicit constraints introduced by the pipeline design of SMT.

Introduction of NMT has considerably changed the field of MT in the recent years. NMT models (such as the ones discussed in Sections 2.3.5 and 2.3.6) surpassed the results of SMT models by a large margin. At the MT shared

task organized by the Workshop on Machine Translation (WMT), nearly all submissions have switched from SMT models to NMT models between 2015 and 2017 [20]. Companies such as Google or Facebook transitioned from SMT to NMT in their translation systems [21, 22].

At the same time, NMT systems have weaknesses that need to be taken into account in practice [23]. Training times of NMT systems can range between days and months even when using GPUs[2]. Large corpora required for training need not to be available for less common language pairs. NMT models are known to make semantical errors such as mistranslation of proper nouns or inconsistencies in numerical expressions; and it is hard to debug the inner workings of a neural network. Also, inference times can be too slow for desktops or mobile phones, which requires the translation systems to be run in cloud environments. Solutions to these problems are subject to ongoing research, with the last problem being also in the focus of this thesis.

2.3.1 Neural Networks

A neural network is a computational model which can be trained to produce output based on given input. It is composed of basic computational units called *neurons*. A neuron receives n input signals $x_1 \dots x_n$ from input data or from some other neurons and computes an output y. Each input signal is multiplied with the weight w_i associated with the i-th input connection. Additionally, there is a constant unit input signal x_{n+1} multiplied by a weight called *bias*. The output of the neuron is the result of the non-linear activation function f applied on the weighted sum of input signals.

$$y = f\left(\sum_{i=1}^{n+1} x_i \cdot w_i\right) \tag{2.6}$$

Neurons in neural networks are organized in *layers*. DNNs contain multiple layers, with *hidden layers* observing only the output of the previous layers. Output of the last layer is the output of the network. Layers of the neural network can be compactly represented as matrices and forward propagation of the signal as matrix multiplication.

The training of neural networks is based on optimization of the weights of the connections between the neurons. The difference between the output of the network and the reference output in the training examples is the *prediction error* of the network. The weights are updated during the process called *back-propagation* to minimize their contribution to the prediction error.

[2]GPU = *graphical processing unit*, a hardware accelerator specialized on numerical computations.

2.3.2 Word Representation

The input of a neural network is a vector of real numbers. In order to process natural language by neural networks, we need a suitable continuous representation of words. A straightforward solution is to use *one-hot* encoding, where each word is associated with a V-dimensional vector (V is the size of the vocabulary, e.g. $V = 50000$). With one-hot encoding, i-th word has 1 on i-th position and 0 on the rest of the positions. For example:

- $aardvark = (1, 0, 0, 0, 0, 0, \ldots)^\top$
- $abacus = (0, 1, 0, 0, 0, 0, \ldots)^\top$.

Another option is to use *word embeddings*. We can find an N-dimensional real-valued vector for each word (where N is low, e.g. $N = 300$) and use this vector as a representation of the word in the continuous space. The aim is to assign similar vectors to similar words. This makes the representation not only compact, but it also allows the neural network to generalize over related words.

One way to define *word similarity* is to assume that similar words occur in similar context. This is used in an approach to learn the word embeddings called *word2vec* [24, 25] in which a neural network with one hidden layer is trained to either predict the current word from its context (continuous bag-of-words) or to predict the context from the current word (continuous skip-gram). The word embeddings are the weights of the hidden layer after the training is finished.

Similar approach is used in end-to-end NMT models for training word embeddings without additional preprocessing steps. The first layer of the model is the *embedding layer* which is designed to learn the word embeddings during the training.

2.3.3 Recurrent Neural Networks

The model of neural networks described in Section 2.3.1 is essentially the *feedforward* model. It has a fixed-size input which limits the size of the context we can feed into the network. Feedforward models (in combination with word embeddings) can be used as an alternative to n-gram models used in SMT [26].

To be able to condition the translation on the input sentence of any length, we can use *recurrent neural networks* (RNNs). RNNs process the input one token at a time, using the same transformation for each step:

$$h_t = A(h_{t-1}, x_t) \tag{2.7}$$

where h_t is the hidden state of the network in time t, A is the transformation applied by the network, and x_t is the t-th input symbol. The transformation is applied repeatedly on the previous hidden state and the current input symbol (see Figure 2.1). After the end of the input is reached, the hidden state should contain an encoded representation of the whole sentence.

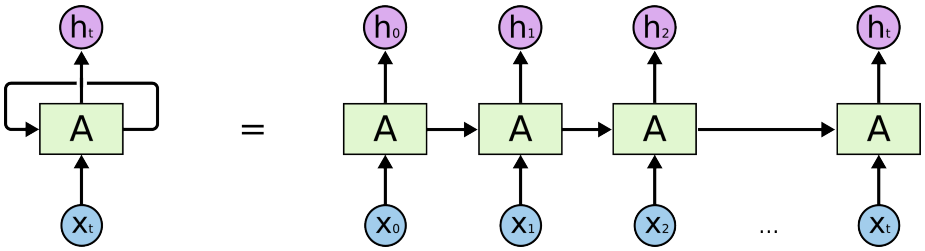

Figure 2.1: Basic scheme of transformations used in an RNN. The scheme on the left shows the transformation applied in each timestep, the scheme on the right shows the same transformation unrolled for each timestep. Source: [27].

In practice, the basic RNN network cannot handle long-distance dependencies so well. As the state gets updated repeatedly, the information from the beginning of the sentence is gradually lost. *Long Short Term Memory* (LTSM) network [28] is a special kind of RNN which is explicitly designed to retain important information from the whole sequence. The transformation is divided between several functional units called *gates* which can learn to filter which information to remember and which to forget. The structure of LSTM is depicted in Figure 2.2.

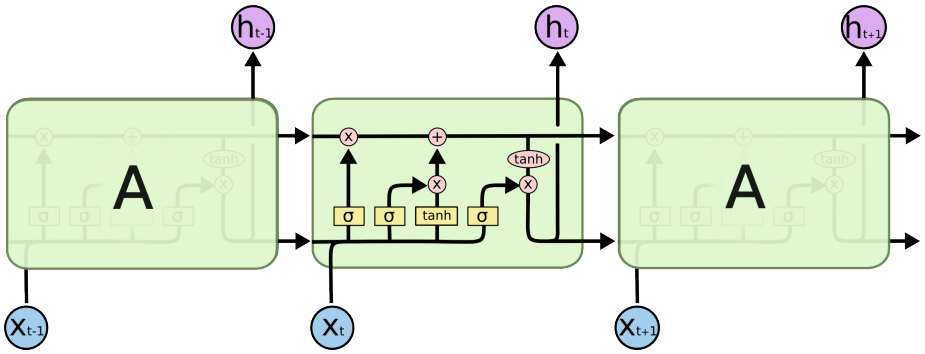

Figure 2.2: Structure of an LTSM. Yellow rectangles are neural network layers, red circles are pointwise operations; together forming the *gates* – the input, forget, and output gate. Source: [27].

2.3.4 Encoder-Decoder Model

Having an efficient way for encoding the information from the sentence into a single vector, we can use another RNN to decode it as another sentence. This is the idea behind the *encoder-decoder* MT model for sequence-to-sequence learning described by *Sutskever et al.* [11] and *Cho et al.* [29] in 2014. The

model operates in two steps:

1. An LSTM called an **encoder** encodes the input sentence $X = x_1 \dots x_n$ into a hidden state h_n by repeatedly applying a transformation E in each timestep $i \in (1, n)$:

$$h_i = E(h_{i-1}, x_i). \tag{2.8}$$

2. An LSTM called a **decoder** uses h_n as its initial state s_0 and produces the translation $Y = y_1 \dots y_m$ of the input sentence by repeatedly applying a transformation D in each timestep $j \in (1, m)$:

$$s_j, y_j = D(s_{j-1}, y_{j-1}). \tag{2.9}$$

The workflow of an encoder-decoder architecture is depicted in Figure 2.3. The encoder takes the input sentence and encodes it into a hidden state. In each timestep, the decoder takes the previous hidden state and the last output symbol, and computes the conditional probability distribution over the vocabulary. The symbol with the maximum probability is selected as the output symbol and used as the input for the next step. The first input symbol is a special token `<s>` denoting the beginning of the sentence. When the decoder outputs a special token `<eos>` denoting the end of the sentence, the decoding is finished.

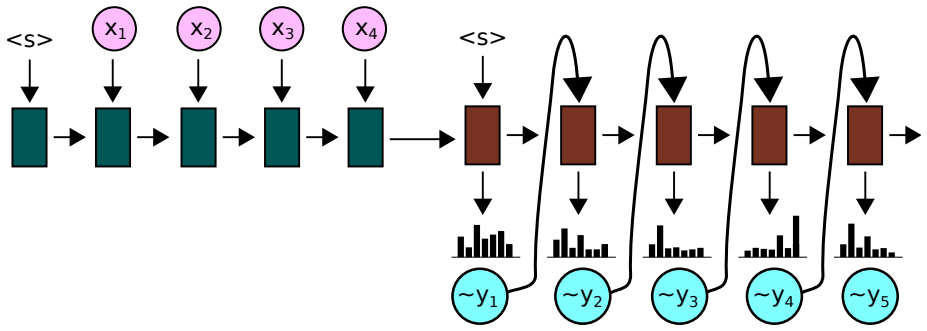

Figure 2.3: *Encoder* encodes the input sentence X into a hidden state, *decoder* uses the hidden state to decode the output sentence Y. Source: [30].

2.3.5 Attention Model

Even LSTMs may not be able to capture all information in the sentences, especially if the sentence is long. The main bottleneck is the fixed-length vector into which all the information has to be compressed. For better flexibility of the encoder-decoder model, *Bahdanau et al.* in 2015 [2] proposed to combine it with the *attention model*. It adds the decoder the ability to extract information from relevant words in the input sentence regardless on their position.

Unlike the standard encoder-decoder model, this approach keeps all the hidden states generated during the encoding process. In each decoding step j, the attention model a computes the vector α_j of attention values for each hidden state h_i of the encoder using softmax:

$$\alpha_{ji} = \frac{\exp(a(s_{j-1}, h_i)}{\sum_k \exp(a(s_{j-1}, h_k))}. \tag{2.10}$$

Figure 2.4 shows how the attention values may be distributed.

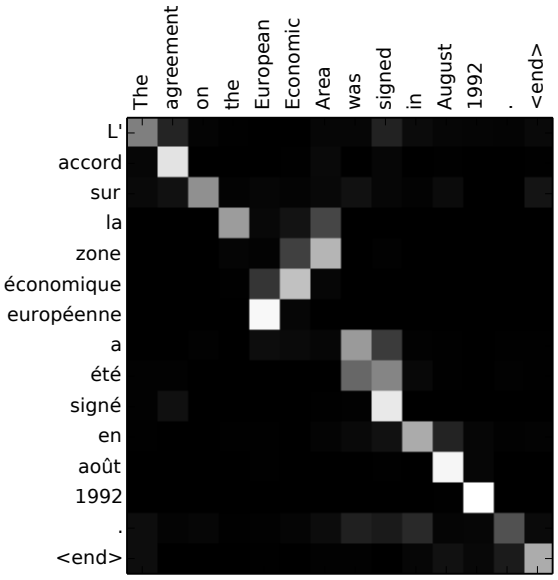

Figure 2.4: Visualization of attention values. Attention model is used for translating the source sentence in English (x-axis) to the target sentence in French (y-axis). Each square shows the attention weight α_{ji} of the i-th source word for the j-th target word, in grayscale (black: 0, white: 1). Source: [2].

The attention vector α_j is used to compute a context vector c_j which is the weighted average of the hidden states of the encoder:

$$c_j = \sum_i \alpha_{ji} h_i. \tag{2.11}$$

The context vector is used as an additional input for the decoder:

$$s_j, y_j = D(s_{j-1}, y_{j-1}, c_j). \tag{2.12}$$

The main difference between the alignment model used in SMT and the attention model used in NMT is that the alignment model is discrete (words are either aligned or not) and the attention model is probabilistic (attention values are real numbers summing up to 1).

2.3.6 Transformer Model

The Transformer model introduced by *Vaswani et al.* in 2016 [31] is the state-of-the-art NMT model which builds on top of the encoder-decoder NMT models with attention. However, it introduces some modifications in the architecture.

Self-Attention. The Transformer completely replaces the RNNs in the encoder and decoder with the mechanism called *self-attention*. The self-attention works similarly to the encoder-decoder attention (see Section 2.3.5), i.e. it computes a single vector as a weighed average of a sequence of vectors. Unlike the encoder-decoder attention, it attends only to the hidden states from the previous layer. In the first layer, the self-attention attends directly to the word embeddings. An example is shown in Figure 2.5.

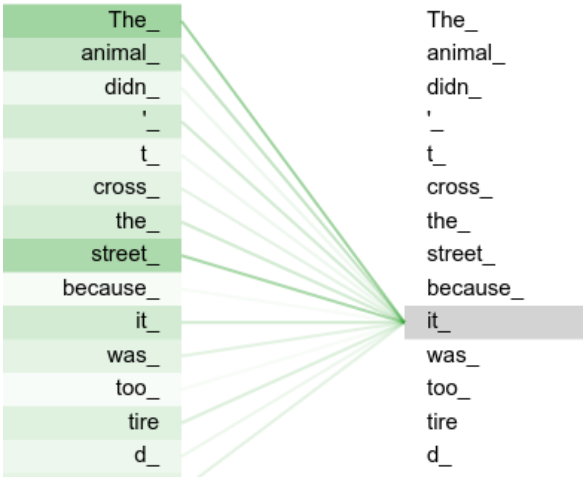

Figure 2.5: Self-attention values of one of the encoder heads while encoding the hidden state for the token `'it_'`. Words with darker color are assigned greater weight. Source: [32].

This leads to the following improvements:

- The model gains an additional *degree of freedom* in encoding the hidden states. It can encode different parts of the sentence into the state and it is not limited to the information from the previous hidden state.
- The computation of the hidden state for each word is *independent* on the computation of the rest of the hidden states. This allows to parallelize the encoding process which is otherwise inherently sequential in RNNs.

Positional Encoding. A side-effect of using the self-attention mechanism is the loss of information about the word order. To address this, the Transformer adds a vector called *positional encoding* to each input word embedding. The

vector is computed deterministically for each position in the sentence. The computation of the vector is designed to encode the relative word order independently on the length of the sentence.

Decoder. The decoder is still *autoregressive*, i.e. conditioned on the previous part of output. During the training, the causality is preserved by masking the future positions in the output so that the decoder can attend only to the decoded part of the sentence.

Architecture. The architecture of the Transformer[3] is shown in Figure 2.6. There are 6 encoder and 6 decoder layers organized in stacks. Each layer is composed of two sublayers: a self-attention sublayer and a single-layer feedforward network. The self-attention sublayer is composed of 8 self-attention instances called *heads* with a different weight matrices, forming a *multi-head attention*. There is a residual connection [33] around each of the two sublayers, followed by layer normalization [34].

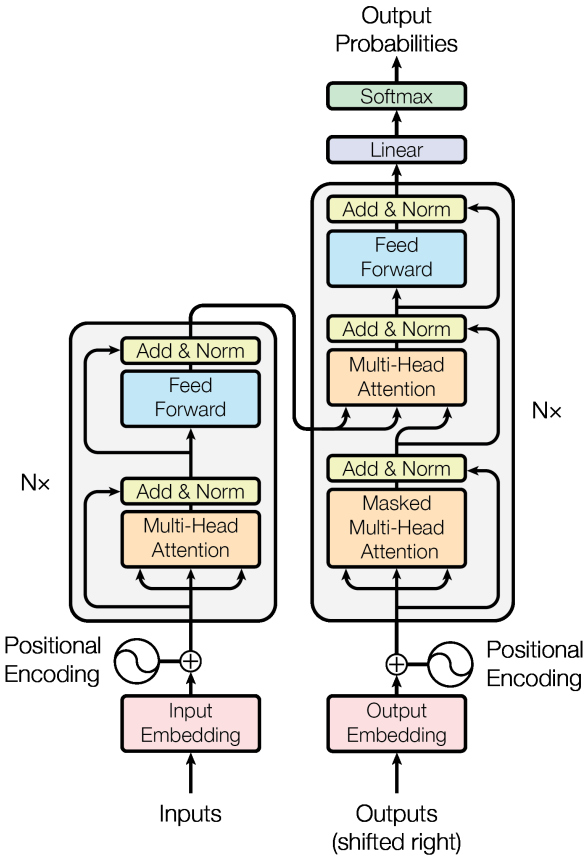

Figure 2.6: The Transformer model architecture. The decoder uses both encoder-decoder attention and self-attention mechanisms. Source: [31]

[3]The architecture with the same hyperparameters is used also in the models building on top of the Transformer.

2.4 Tokenization

A *token* (resp. a *subword*) is a sequence of characters which is treated by the MT model as a single and indivisible item in the vocabulary. *Tokenization* (resp. *segmentation*) is a process of breaking the input text into tokens[4].

In SMT, tokenization is traditionally used as a part of a pipeline. Although NMT models can be trained end-to-end without any preprocessing or post-processing, good tokenization helps the models to deal with out-of-vocabulary words while keeping the size of the vocabulary reasonably large. This is important as machine translation is an open-vocabulary problem. During *detokenization*, tokens can be composed into words which did not appear in the training data, e.g. in case of translating proper names or compound nouns.

As an example of advanced tokenization technique for NMT, the authors of [35] propose encoding words into subwords using the *byte pair encoding* (BPE) compression algorithm [36]. BPE gets a list of subwords by splitting the sentence into individual characters and consecutively merging the most frequent adjacent pairs of characters, until the desired vocabulary size is reached. Tokenization is achieved by applying the same merge operations to the input text. Another tokenization algorithm proposed in [37] is the *unigram language model* which is capable of outputting multiple subword segmentations with probabilities.

2.5 Beam Search

In theory, the output of the decoding algorithm should be the most probable target translation, i.e.

$$Y^* = \arg\max_Y p(Y|X). \tag{2.13}$$

However, the search problem is exponential in the length of the output sequence which is intractable in practice.

Approximate decoding algorithms were developed to deal with this issue. The most straightforward approach is to use the *greedy decoding* algorithm, i.e. in each step, output a token with the highest probability. Note that this does not implicitly maximize the probability of the whole output sequence, as the output in each step is conditioned on the output from the previous steps.

An extension of the greedy decoding algorithm is the *beam search* algorithm. The idea is to keep b partial outputs called *hypotheses*. In each step, b best

[4]Traditionally, tokenization and tokens are based on syntax while segmentation and subwords are based on semantics. As the tools described in this thesis work somewhere in between of those approaches, we will use the terms interchangeably.

hypotheses are extended by an output token, resulting in $b \cdot V$ new hypotheses (V is the size of the vocabulary). A score is computed for each hypothesis as the product of probability of its tokens normalized by the hypothesis length. The hypotheses are re-ranked by their score and b hypotheses with the highest score are kept for the next decoding step.

Beam search allows to account for more possible translations, which improves the results of the decoding algorithm. It introduces some computational overhead, which can be adjusted by altering the beam size b. In SMT, large beam sizes ($b = 1000$) are used [38]. In NMT, $b = 10$ can be enough for a good approximation [39]. With $b = 1$, this approach effectively becomes the greedy decoding algorithm.

2.6 Evaluation Metrics

A standardized metric for evaluating the results of MT algorithms is not easy to define – a single text may have multiple correct translations and their correctness may be subjective. It is also not straightforward how to define how far is an *incorrect* translation from the *correct* one.

Initially, most of the measures of MT quality were performed by humans. People were asked to evaluate various aspects of the translated sentence such as comprehensibility, fluency, or accuracy [9]. However, manual evaluation is expensive and time-consuming, and it is in conflict with the aim to reduce the dependency of MT on human input. With this in mind, automated evaluation metrics were developed.

2.6.1 BLEU

BLEU (Bilingual Evaluation Understudy) score introduced in [40] is an inexpensive and language-independent method for automatic MT evaluation. It builds the intuition on a principle *the closer a machine-generated translation is to a professional human translation, the better it is.* The *closeness* is computed as a weighted average over different n's of n-gram matches against the reference translation, combined with a brevity penalty for short sentences.

Formally, BLEU score is computed as

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{2.14}$$

where p_n the number of n-grams in the candidate translation present in the reference translation, divided by the total number of n-grams in the reference translation; w_n are positive weights summing to one; and BP is the brevity

penalty computed as

$$\mathrm{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{otherwise,} \end{cases} \tag{2.15}$$

where c is the length of the candidate translation and r is the length of the reference translation.

BLEU score is used for evaluation of translation quality for the experiments in this work as the de facto standard metric for MT evaluation [41].

2.6.2 Translation Edit Rate

An alternative metric proposed in [42] is the Translation Edit Rate (TER), which is based on the Levenshtein distance. TER measures the minimum number of edits that a human would have to perform to change a system output so that it exactly matches a reference translation. The edits include an insertion, deletion or substitution of a word, and an edit which moves sequences of contiguous words. TER is computed as:

$$\mathrm{TER} = \frac{1}{r} \cdot d_L(t_r, t_c) \tag{2.16}$$

where $d_L(t_r, t_c)$ is the Levenshtein distance between the reference and candidate translations, and r is the length of the reference translation.

Chapter 3

Non-autoregressive MT

This chapter presents the motivation behind the non-autoregressive MT and the work that has been done in this field so far. The approaches which are presented also include the end-to-end model with CTC, which is used as a base model in this thesis.

3.1 Motivation

Despite the advances of NMT in translation quality, NMT models are *slow* during the inference time. This prevents deployment of high-quality translation systems into offline desktop or mobile environment.

There are two main reasons for the low inference speed:

1. DNNs are computationally **expensive to evaluate**. In case the expensive evaluation is combined with linear complexity of the algorithm, it can result in low inference speed for longer sentences.

2. Some parts of the models are **not parallelizable**. Parallelization can help to improve the speed of evaluation of DNNs, but if some part of the model is inherently sequential, it slows the algorithm down in accordance with the Amdahl's law [43].

There are efforts to solve these problems since the introduction of the first NMT models. Almost every part of the Transformer model (Section 2.3.6) can be parallelized and it is more economical in the number of operations compared to other models [31, 44].

However, the autoregressive decoder used in all the NMT models described in Section 2.3 (including the Transformer) cannot be parallelized. In every step, the output of the decoder depends on its output from the previous step (hence *autoregressive*). This makes the decoder inherently sequential.

The autoregressivity has several benefits. It corresponds to the sequential nature of language and the way the real translations are produced. Autoregressive models achieve state-of-the-art performance on large corpora and are easy to train.

Nevertheless, as autoregressive decoders became the main bottleneck of speeding up the NMT models, recent works have attempted to solve the issues by the **non-autoregressive** approach. In non-autoregressive models, the conditional probability of the target sentence is factorized, making the probability of each output token independent on the rest of the sentence. In other words, the probability distribution of each token y_j depends only on the source sentence X.

Intuitively, this naive approach does not yield good results. The main issue described in [44] is the *multimodality problem*. There may be more acceptable translations for the target sentence, forming a probability distribution. As each of the target tokens draws from the distribution independently, it may happen that a different translation is selected for each token. An example may be the English sentence *"Thank you."*, which can be translated into German as *"Danke."*, *"Danke schön."* or *"Vielen Dank"*. As the non-autoregressive model assumes the probability distributions for each word are conditionally independent, the model can also assign high probability to translations *"Danke Dank."* and *"Vielen schön."*, which are not correct.

Another problem is the *unknown length* of the target sentence prior to translation. As all output tokens are decoded independently and in parallel, we cannot use the `<eos>` token to terminate the decoding and we have to estimate the target sentence length in advance.

The models described in the following sections use various approaches to tackle these drawbacks. All the models build on top of the autoregressive Transformer model with self-attention and introduce modifications for non-autoregressive translation.

3.2 Latent Fertility Model

The first non-autoregressive NMT model was described by *Gu et al.* in 2017 [44]. The Transformer encoder part stays unchanged, but the decoder part is modified:

- **Decoder input**. As the decoder is non-autoregressive, it can no longer use previously predicted output as its input. However, omitting the input for the decoder entirely leads to poor performance, so it is instead substituted by copied encoder input (details are discussed below).
- **Causality mask**. There is no need to prevent earlier decoding steps from accessing information from later steps, so the mask from Transformer is no longer used for future positions in the output during the training.

- **Positional attention**. Decoder is equipped with additional positional attention module. It acts similarly to self-attention, but instead of word embeddings it attends to positional encodings. Positional attention should help the decoder to perform local word reordering.

The model introduces an additional step to the processing pipeline. A latent *fertility model* is introduced to estimate the fertility (see Section 2.2.2) of each word. Based on their fertility, the words from the source sentence are repeated in the decoder input. The resulting output length is determined by the sum of all fertility values.

The latent fertility model aims to remove the non-determinism in the decoding process in order to tackle the multimodality problem. By predicting the fertility sequence, it introduces a latent variable z. Conditioning the translation on z should filter out the target sentences which are not consistent with the particular fertility sequence.

Using only the fertilities is still not sufficient to solve all cases of the multimodality problem. In many cases, there are multiple correct translations consistent with the same sequence of fertilities. The authors additionally try to apply sequence-level knowledge distillation [45] to rescore the possible output sentences using an autoregressive model. They note that this approach makes the resulting translation less noisy and more deterministic, but lower in quality.

3.3 Model with Iterative Refining

Following up on the latent fertility model, *Lee et al.* in 2018 [46] introduced a non-autoregressive NMT model based on iterative refining. The hyperparameters are similar to the previous model, but there are several changes in the model architecture.

Instead of estimating the fertility of each word, the model estimates only the total length of the target sentence. It is predicted by a separate network which receives the output of the encoder on its input. This is a fast and simple way to help the model in decoding the target sequence, however, it leaves the task of improving translation quality up to another part of the model.

The translation quality is improved by a second decoder. It is applied repeatedly on the candidate translation that is produced by the first decoder. This process is called *iterative refining*. The second decoder acts as a denoising autoencoder – it views the candidate translation as a corrupted version of the correct output and it tries to restore it. The decoder is trained separately from the rest of the model.

3.4 End-to-End Model with CTC

The model introduced by *Libovický and Helcl* in 2018 [4] eliminates the need for explicit target sentence length estimation by using the CTC algorithm. The model can be trained end-to-end and it is used as a base model for the work in this thesis.

3.4.1 Connectionist Temporal Classification

Connectionist Temporal Classification (CTC) [47, 48] is an algorithm dealing with alignment of two sequences.

The problem is formulated as follows: we are given an input sequence $X = (x_1, \ldots, x_m)$ and an output sequence $Y = (y_1, \ldots, y_n)$. We want to find the alignment between the sequences, i.e. a mapping of elements of X to the elements of Y, given that:

- The input sequence X may be longer than Y (i.e. $m \geq n$).
- The correspondence between the elements of X and Y is not known.
- The alignment is monotonic: if $x_i \in X$, $y_j \in Y$ and x_i is aligned with y_j, then x_{i+1} is aligned either with y_j or with y_{j+1}.

The problem can be illustrated on an example from speech recognition. X is the audio signal split into m samples, Y is the output sequence composed of n characters. We do not know n in advance, but we know that each character corresponds to one or more samples. A possible way to perform the alignment is to assign a character to each sample and collapse the repeats. In order to be able to decode multiple same characters in a row, we introduce a special character ϵ which acts as a delimiter. See Figure 3.1.

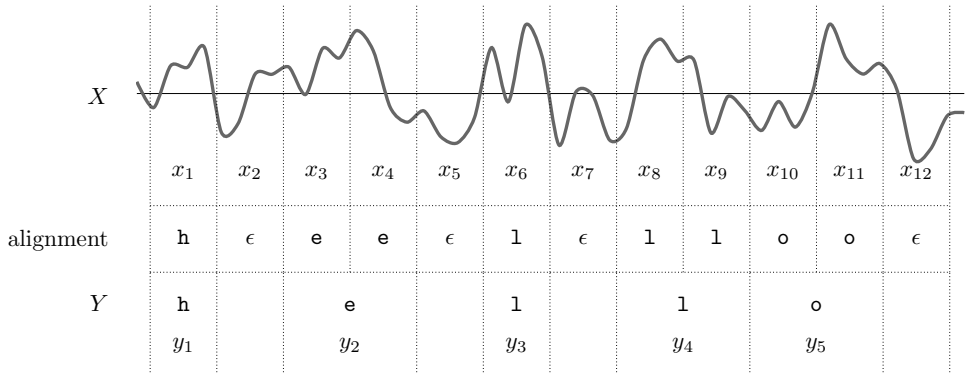

Figure 3.1: A example of an alignment between the audio wave samples X and the sequence of characters $Y =$ `hello`.

Multiple alignments may exist for a single pair of X and Y. For example, the following alignments are all valid for the input sequence X of length $n = 6$ and the output sequence $Y =$ `cat`:

- `(c, a, a, t, t, t)`
- `(c, c, a, ϵ, ϵ, t)`
- `(c, ϵ, a, a, t, t)`.

CTC algorithm gives us an efficient way to compute the distribution of all possible Y's for a given X, marginalizing over all possible alignments for each Y. This can be used for computing:

- **output** – during the inference, we can compute $Y^* = \arg\max_Y p(Y|X)$ to get the most likely output sequence Y given the input sequence X
- **loss function** – during the training, we can compute $p(Y|X)$ for the output sequence Y from the model for the reference input sequence X.

The use of CTC for computing the output is described in detail in Section 4.1. The loss function is computed as a sum of probabilities over the possible alignments, each probability calculated as a product of the probabilities of the alignment a_t given the input sequence X in each timestep $t \in T$:

$$p(Y|X) = \sum_{a \in A} \prod_{t=1}^{T} p_t(a_t|X). \tag{3.1}$$

Computing the probability for all the possible alignments one-by-one would lead to a combinatorial explosion. For example, for X of length $m = 50$ without any repeating elements and Y of length $n = 30$, the number of possible alignments is

$$\binom{m+n}{m-n} = \binom{80}{20} \approx 3 \cdot 10^{18}. \tag{3.2}$$

To avoid this problem, CTC uses *dynamic programming*. It builds on the insight that if two alignments have reached the same output at the same step, they can be merged. Let $Z = (\epsilon, y_1, \epsilon, \ldots, \epsilon, y_n, \epsilon)$ be a sequence containing elements of Y with ϵ in between each two elements (and at the beginning and the end of the sequence). We denote the CTC score of the subsequence $Z_{1:s}$ after t timesteps as $\alpha_{s,t}$. We have two cases for computing $\alpha_{s,t}$:

1. In this case $z_s \in Y$ and $z_s \neq z_{s-2}$. This means we have three possible positions in the previous step that we have to consider:

$$\alpha_{s,t} = (\alpha_{s-2,t-1} + \alpha_{s-1,t-1} + \alpha_{s,t-1}) \cdot p_t(z_s|X). \tag{3.3}$$

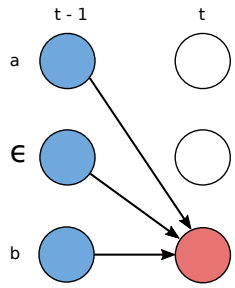

Figure 3.2: Case 1. We consider three possibilities in the previous timestep: previous character `a`, delimiter ϵ, or current character `b`. Source: [48]

2. In this case either $z_s = \epsilon$ or $z_s = z_{s-2}$. This means that we cannot skip the previous token in Z. If $z_s = \epsilon$, it is because we cannot skip over any elements of Y (and $z_{s-1} \in Y$); if $z_s = z_{s-2}$, it is because we have to keep ϵ between two identical characters. Therefore:

$$\alpha_{s,t} = (\alpha_{s-1,t-1} + \alpha_{s,t-1}) \cdot p_t(z_s|X). \tag{3.4}$$

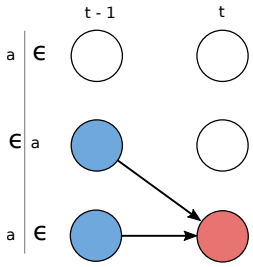

Figure 3.3: Case 2. We consider two possibilities in the previous timestep: previous character ϵ (resp. `a`), or current character `a` (resp. ϵ). Source: [48]

Using these two steps, we can compute paths for all possible alignments and use the score from the two final nodes[1] as $p(Y|X)$ (see Figure 3.4).

3.4.2 Model Architecture

The model [4] follows the architecture of the Transformer and the other non-autoregressive models. However, it does not estimate the target sentence length. Instead, it only sets its *upper bound* as k-times the length of the input sentence (in the experiments $k = 3$).

This allows to decode a sentence of any length (shorter than the upper bound) in the following way: after encoding the input sequence X of length

[1] The delimiter ϵ at the end of the alignment is optional, so we have to sum the score from both nodes to get the final score.

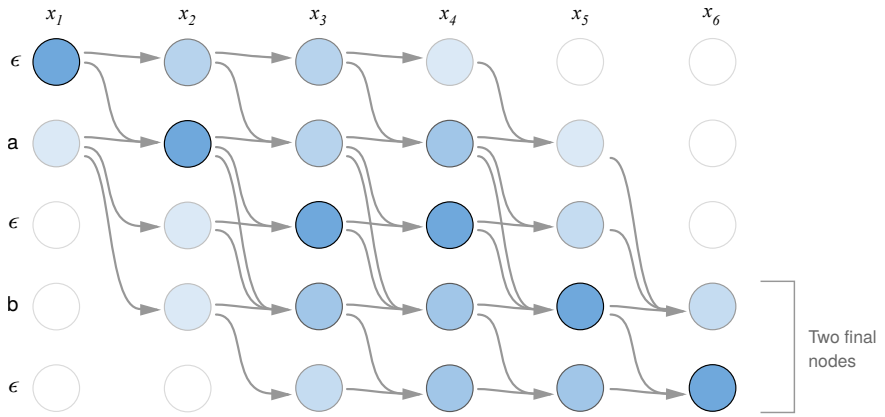

Figure 3.4: Input $X = (x_1, \ldots, x_6)$, output $Y =$ `ab`. Each node (s, t) represents $\alpha_{s,t}$ – the CTC score of the subsequence $Z_{1:s}$ after t steps. Source: [48]

n, each encoder state is projected to k vectors, forming a sequence of length $k \cdot n$. This sequence is processed by the decoder and labeled either with an output token or with the ϵ symbol.

The labeling can be processed by the CTC algorithm similarly to the example with the audio signal shown in Figure 3.1. The ϵ tokens act as delimiters and mark the decoder states which do not produce any word during the decoding process.

Figure 3.5 shows the architecture of the model.

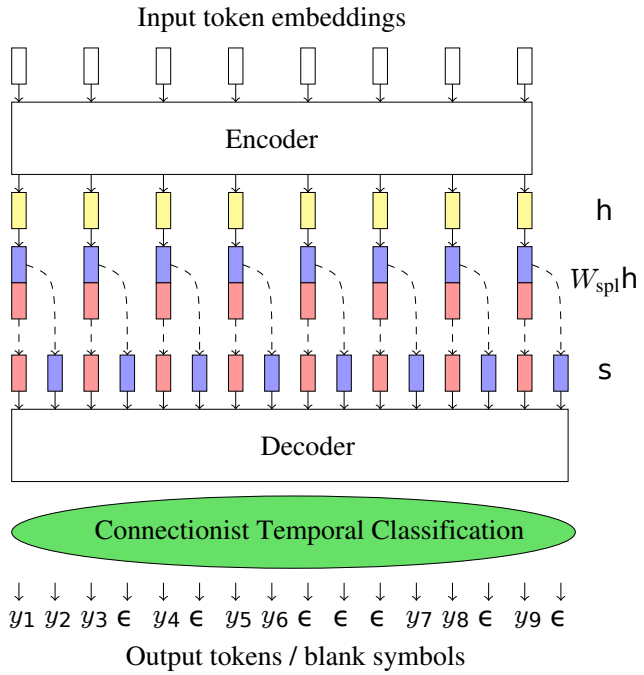

Figure 3.5: The architecture of the end-to-end model with CTC. Source: [4].

Chapter 4

Design

This chapter describes the main contribution of this thesis – a novel approach for decoding the target translation in a non-autoregressive NMT model. The model builds on top of the end-to-end model with CTC described in Section 3.4. The decoding process is improved by a scoring model which uses a linear combination of feature values to score the translation hypotheses. The weights of the features are determined using a variant of a structured perceptron algorithm.

4.1 Beam Search with CTC

The beam search algorithm (described in Section 2.5) does not bring any advantage to the non-autoregressive models described in Sections 3.2 and 3.3. By definition, the output tokens in these models are mutually independent and selecting the token with maximal probability also maximizes the probability of the whole sentence. This allows to use greedy decoding and to parallelize the decoding process.

In the end-to-end model with CTC from Section 3.4, however, there may be multiple derivations of the same hypothesis, i.e. yielding the same output sentence (see Section 3.4.1). In each decoding step, we have to combine probabilities of all the derivations of the same hypothesis. This implies that the inference algorithm in this model is sequential and its runtime is linear in the length of the output sentence, although it is still faster than the autoregressive models[1]. In this case, the beam search algorithm can be used for improving the quality of the translation as the greedy algorithm is not guaranteed to find the sentence with maximal probability.

We use this property of the model as an opportunity to introduce a more

[1]It is important to note that the model stays non-autoregressive in a sense that it still requires only a **single pass** through the decoder stack, i.e. the token probability distributions for each output position are computed in parallel. The only sequential part of the process is combining the computed probabilites, which is fast in practice.

complex scoring model for the beam search. Instead of computing the probabilities of translation hypotheses using only the CTC score, we compute the score of the hypotheses as a linear combination of features.

Algorithm 1 and Figure 4.1 provide an overview of the decoding algorithm. The algorithm runs in $k \cdot T_X$ steps, where T_X is the length of the input sentence X and k is the multiplication factor of the encoder states. The derivations from the previous steps are expanded with $2b$ tokens with the highest score[2]. The score of a single derivation is the product of the conditionally independent probabilities of its tokens (line 7). The CTC score of a hypothesis is the sum of the scores of its derivations formed in the current beam search step (line 8). The function SelectNBest is based on the scoring model described in Section 4.2.

Algorithm 1 Beam Search Algorithm with CTC

```
1:  B ← {∅}                                              ▷ Beam
2:  for t = 1 to k · T_X do
3:      H ← ∅                                            ▷ Hypothesis → CTC score
4:      W ← 2b best tokens in step t
5:      for hypothesis h ∈ B do
6:          for token w ∈ W do
7:              s ← p(h) · p_t(w)                        ▷ Derivation score
8:              H[h + w] ← H[h + w] + s
9:          end for
10:     end for
11:     B ← SelectNBest(H, b)
12: end for
13: return B
```

4.2 Scoring Model

The function SelectNBest(H, n) (line 11 in Algorithm 1) selects n hypotheses from H with the highest score. We employ a linear model for scoring the hypotheses. We compute the score of the hypothesis h as

$$score(h) = \mathbf{w} \cdot \mathbf{\Phi}(h) \tag{4.1}$$

where $\mathbf{\Phi}$ is a feature function of h and $\mathbf{w}$ is a feature weight vector. The feature function $\mathbf{\Phi} : \mathcal{H} \to \mathbb{R}^4$ maps a derivation of a hypothesis to an n-dimensional vector, where n is the number of features. In our case

$$\mathbf{\Phi}(h) = (\Phi_c(h), \Phi_l(h), \Phi_r(h), \Phi_t(h)). \tag{4.2}$$

The individual features are described in the following sections.

[2] In contrast to the standard beam search formulation, we use only by $2b$ tokens with highest score, as expansion by all V tokens would prohibitively slow down the algorithm and increasing the coefficient did not bring any improvement in our experiments.

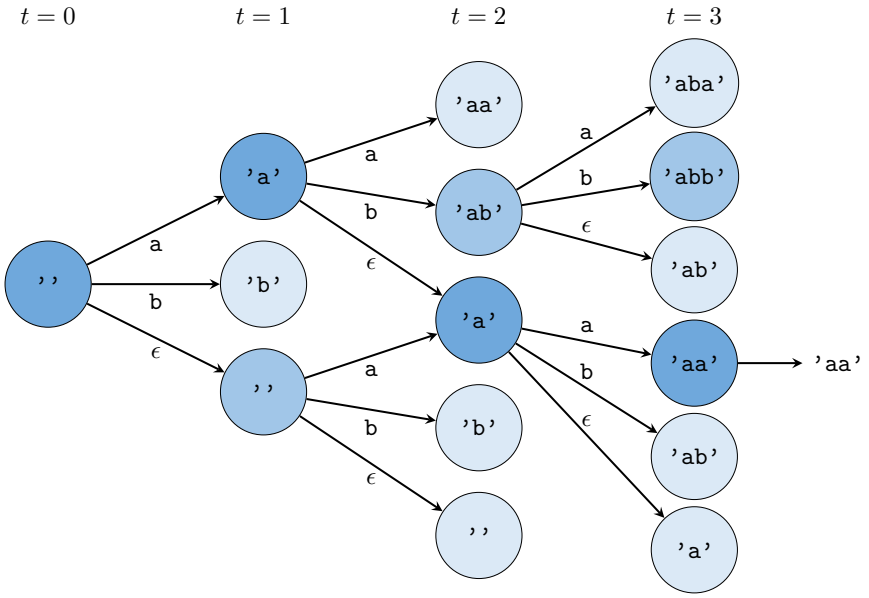

Figure 4.1: An example of the beam search with CTC: vocabulary = $(\mathtt{a}, \mathtt{b}, \epsilon)$, $b = 2$, $k \cdot T_x = 3$. In each timestep t, b best hypotheses from the previous step are expanded with the tokens from the vocabulary and ranked by their score from the scoring model (darker color indicates higher score). The scores of different derivations of a single hypothesis are combined. The resulting sentence after $k \cdot T_x$ steps is `'aa'` of length 2.

4.2.1 CTC Score

We use the CTC score from the original end-to-end model as one of the features. We work with log-probabilities to avoid numerical issues, i.e.

$$\Phi_c(h) = \log p(h|X). \tag{4.3}$$

It is important to note that unlike other features, we *do not train* the weight for this feature. Instead we set it to 1 in order to normalize the vector as the linear scoring model is independent on the scale s of the vector:

$$\mathbf{w} \cdot \mathbf{\Phi}(h) \approx s\mathbf{w} \cdot \mathbf{\Phi}(h), \quad s \in \mathbb{R}_{>0}. \tag{4.4}$$

4.2.2 Language Model

In SMT, an LM is responsible for handling the fluency of the translation (see Section 2.2.1). There is no LM in autoregressive NMT – the NMT model is trained end-to-end and the decoder part plays a role of a conditional LM.

In non-autoregressive NMT, the output tokens are not conditionally dependent on the rest of the output and the LM component is missing. Therefore, the translations may be less fluent – the words are unnecessarily repeated, the

word order is incorrect and the sentences are generally less comprehensible (regardless of the adequacy of the translation given the input sentence).

Previous research indicates that incorporating a LM component into the decoding process can leverage the quality of results. In [49], the authors experiment with using a LM trained on target monolingual data to improve the quality of low-resource language pairs (with not enough parallel data) and show improvement of the results of the NMT model from [2] on both low-resource and high-resource language pairs. In automatic speech recognition, the authors of [50] propose a beam search algorithm which combines an n-gram LM with scores from a model trained using CTC and achieve state-of-the-art accuracy.

Following up on the research, we also use an n-gram LM for scoring the hypotheses. Similarly to more complex approaches like recurrent LMs, which take into account the whole output sentence, an n-gram LM is able to assign low-probability to repeating words or words that do not belong together in the target language and thus handicap derivations containing these mistakes. Unlike those approaches, an n-gram LM is not too computationally demanding which allows to maintain the decoding speed.

In particular, we use KenLM [51] described in Section 5.3. The score of a sentence in KenLM is based on a product of probabilities of n-grams in the sentence, similarly to Equation 2.5. However, KenLM uses modified Kneser-Ney smoothing [52], which is an interpolation technique allowing better estimation of n-gram probabilities. The main idea (out of several) in this technique is using a *back-off model* for calculating the score of a single n-gram. If the n-gram did not appear in the data, we use a probability of a lower order n-gram together with a backoff penalty b:

$$p(w_n|w_1^{n-1}) = p(w_n|w_f^{n-1}) \prod_{i=1}^{f-1} b(w_i^{n-1}), \tag{4.5}$$

where $w_1^n = (w_1, \ldots, w_n)$ is an n-gram and w_f^n is the longest part of the n-gram encountered in the data.

The value of the LM feature in the scoring model is based on the log-probability of the sentence as calculated by KenLM. Since the hypotheses may contain blank symbols, the beam may consist of hypotheses of different lengths. Because shorter sequences are favored by KenLM, we divide the log-probability of each hypothesis h by its length l in order to normalize the scores:

$$\Phi_l(h) = \frac{LM_n(h)}{l}. \tag{4.6}$$

4.2.3 Blank / Non-Blank Symbol Ratio

Besides the impaired fluency, translations from the non-autoregressive model often suffer from output that is too short compared to the reference output.

This motivated us to design features for the linear model penalizing the hypotheses with short output.

The number of decoder states is equal to the number of encoder states multiplied by factor k. Intuitively, we would expect that on average, for every sequence of decoder states of length k, there should be approximately one decoded *non-blank* symbol (i.e. an output token) and $k-1$ *blank* symbols in order to maintain the length of the sentence. In practice, this estimate may differ depending on:

- **language pair** – the languages may have sentences of different average length
- **input-output sentence pair** – the length of the translation may differ from the length of the input sentence
- **position in the decoded sentence** – the ratio fluctuates more when there are less decoded tokens.

Nevertheless, adding a feature based on this ratio can inform the model that there are too many (or too few) blank symbols in the output and guide the decoding toward sentences of correct length. We formulate the feature as

$$\Phi_r(h) = \max\left(0, \frac{\#\text{blanks}(h)}{\#\text{non-blanks}(h)} - \delta\right) \tag{4.7}$$

where δ is a hyperparameter that thresholds the penalization for the ratio. In other words, the feature penalizes the hypothesis if the ratio is higher than δ and it stays zero otherwise, which allows the ratio to vary to certain extent.

To estimate the penalization threshold δ, we observed how the ratio behaves during the decoding. We used the model with the splitting factor $k = 3$ trained for the English-German language pair. We ran the model on the validation dataset and for every input sentence, we employed dynamic programming to find the most likely derivation of the reference translation – see the function FindCtcHypothesis in Algorithm 3 in Section 4.3.2. For every timestep in the derivation, we computed the ratio of blank and non-blank symbols. The histogram of values of the ratio (up to 10) is shown in Figure 4.2.

We can see that the empirical distribution has a peak at $k-1=2$, which confirms the previous intuition. Furthermore, most of the distribution mass lies in the interval $(0, 4) = (0, 2 \cdot (k-1))$. We make an assumption that a ratio above $2 \cdot (k-1)$ is an outlier and the derivation with such ratio should be penalized. Therefore, in our experiments we set $\delta = 4$.[3]

Note that if the ratio is lower than 1, it signalizes that the output could be too long. Nevertheless, this kind of misbehavior is rare with our datasets and we decided not to penalize this case.

[3]The languages in our experiments have sentences of similar length. For more distant language pairs (e.g. English-Chinese), the value of δ should be reconsidered.

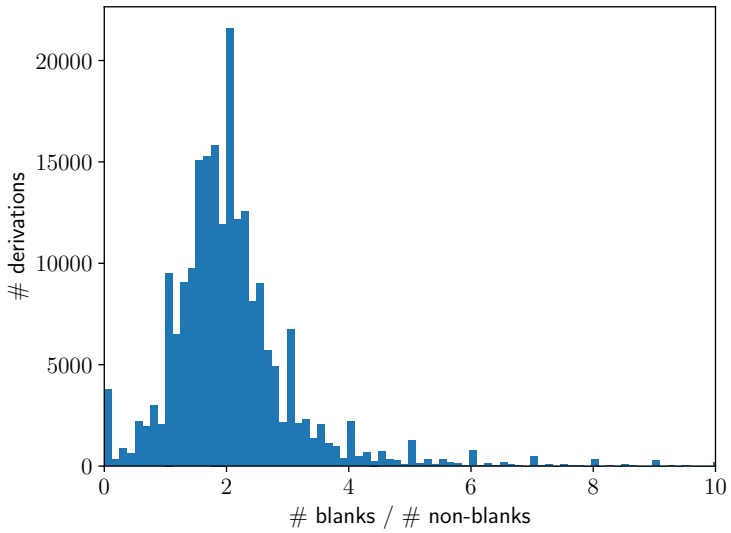

Figure 4.2: Histogram of values of blank/non-blank ratio for the model with $k = 3$. The ratio was computed at each timestep from the most likely derivations of reference translations on the en-de validation dataset. The peak is centered around $k - 1 = 2$, i.e. most commonly there are two blank symbols for each non-blank symbol in the decoded hypothesis.

4.2.4 Trailing Blank Symbols

Yet another case which violates the intuition behind the correct decoding process is a *long sequence of blank symbols* in the derivation. This can be another indication that the target translation will be too short. As the non-blank symbols from the beginning of the hypothesis may prevent the blank/non-blank ratio to control this aspect, we add a feature counting the number of trailing blank symbols:

$$\Phi_t(h) = \max\left(0, \#\text{ trailing blanks} - \gamma(X)\right). \tag{4.8}$$

We assume that more blanks in a row can appear during the decoding of longer sentences. Therefore, the normalization coefficient γ is a function of the input sentence X (as we do not have the output sentence Y at the time of inference). We used the similar approach to estimate the function, i.e. we used the model with $k = 3$ trained for the English-German language pair and we used the function FindCtcHypothesis to find the most likely derivations of the reference translations on the validation dataset. Figure 4.3 shows maximal lengths of a sequence of blank symbols in the derivation of the reference translation depending on the number of the input sequence tokens.

We can see that $\gamma(X) = \#\text{ tokens}(X)$ is a reasonable estimate for the threshold of the longest sequence of blank symbols during the decoding.

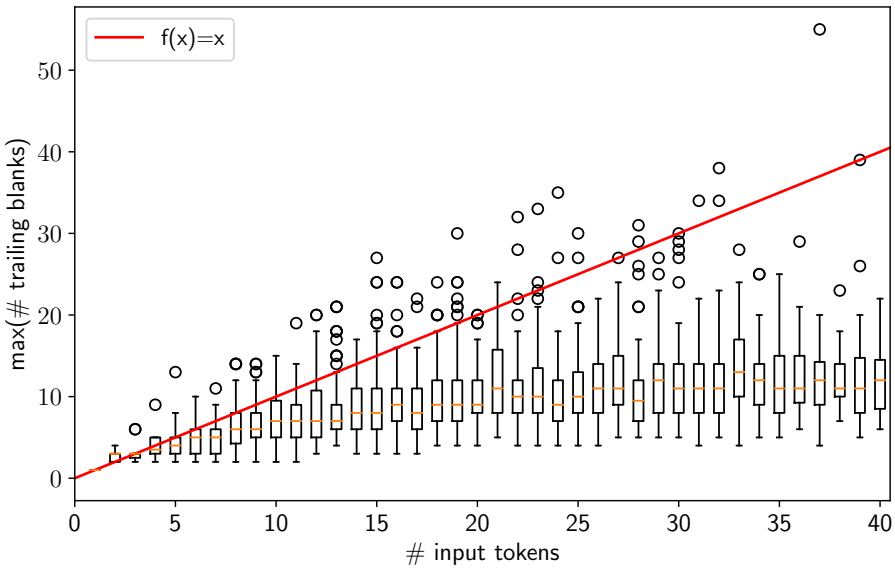

Figure 4.3: A boxplot for every number of input sentence tokens. Each boxplot shows the quartiles of the distribution of the maximal length of a sequence of blank symbols in the derivation of the reference translation. The red line shows the graph of the function $f(x) = x$ used for estimating the normalization coefficient γ.

Although the estimate could be adjusted for longer sentences, we decided to keep this version in our experiments for simplicity.

4.3 Feature Weights

The score of a hypothesis is computed as a linear combination of its features. The weights for the linear combination are stored in the vector $\mathbf{w}$, which is a hyperparameter of the neural model.

The importance of each feature (which should be reflected in its weight) can differ for each language pair and for each trained instance of the model. As we have no prior assumptions about the features, we need a reliable and automatic way to estimate the value of this hyperparameter.

4.3.1 Brute-Force Search

We can find the value of $\mathbf{w}$ with some kind of brute-force search technique, e.g. grid search or random search. In *grid search*, we create a list of values (usually equidistant) for each weight and perform an exhaustive search over all the possible combinations. Similar approach is *random search* in which we also try all the combinations, but the values for each weight come from uniform

random sampling. This was shown to be more efficient for hyperparameter optimization than grid search [53].

However, neither of the approaches is reasonably efficient in our case. As we do not have any prior information about suitable values, we would need many samples to cover a wide range while keeping relatively high density of samples. This would quickly lead to combinatorial explosion – for 50 samples and 3 features we would need $50^3 = 125\,000$ trials to test all the possible combinations.

4.3.2 Structured Perceptron

Our problem is a sequence labeling task – we want to find a sequence of labels h for a sequence of hidden states conditioned on the input sequence X using a linear model s.t. the score in Equation 4.1 is maximized. Machine learning techniques were developed for this kind of task (e.g. conditional random fields [54], structured perceptron [55], structural SVMs [56]). Learning the weights can help to combat the inefficiency of the brute-force search. Nevertheless, all these techniques assume *exact inference* to guarantee their theoretical convergence properties. As we use beam search for the decoding, we have to use a technique which supports inexact inference instead.

A variant of structured perceptron for inexact search was suggested in [5]. The authors develop a theoretical framework of a *violation-fixing* perceptron, which is guaranteed to converge even with inexact search, given that each perceptron update contains a violation. A *violation* is an example with higher score than the correct example. As we will describe later, we can use a hypothesis that stays in the beam after the reference hypothesis falls off the beam as a violation.

Algorithm 2 describes the variant of the structured perceptron we used for learning the value of $\mathbf{w}$. The algorithm operates on dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ which is a set of N pairs, each pair consisting of a sequence of hidden states x (the output from the decoder after the model processes the corresponding input sentence X) and the reference translation y. For every example in D, we use a function FindCtcHypothesis to find the most likely hypothesis h for the reference translation y and the function FindViolation to find a violation $\hat{h}$. We update the vector $\mathbf{w}$ by the difference in features between h and $\hat{h}$. We use the learning rate α to adjust the size of the update.

The function FindCtcHypothesis in Algorithm 3 allows us to find the most likely derivation h of the reference translation y. This is necessary because y may have many possible derivations and the features are computed for a particular derivation. The algorithm uses dynamic programming to find the most likely path to decode y. We know that at each timestep, the derivation of y can be expanded either by ϵ or by the next token of y. We also know that in the last timestep, h has to contain all the tokens of y. We fill the possible derivations in the table A, in each timestep expanding the derivation

Algorithm 2 Violation-Fixing Structured Perceptron

$D \leftarrow \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$
$\mathbf{w} \leftarrow \mathbf{0}$
repeat
  for each $(x, y) \in D$ **do**
    $h \leftarrow \text{FindCtcHypothesis}(x, y)$ ▷ Algorithm 3
    $(\hat{h}, t) \leftarrow \text{FindViolation}(x, y, \mathbf{w})$ ▷ Algorithm 4
    if $h \neq \hat{h}$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\mathbf{\Phi}(h_{1:t}) - \mathbf{\Phi}(\hat{h}))$
    end if
  end for
until converged

from the previous timestep and setting the parent pointer accordingly. If there are more possible derivations for a single timestep and a number of expanded tokens, we keep the derivation with higher CTC score. After A is filled, the path is reconstructed by following the parent pointers.

Algorithm 3 Most Likely Derivation of Reference Translation Using CTC Score

function FindCtcHypothesis(x, y)
  $A \leftarrow$ array $T_y \times T_x$ ▷ T = length of the sequence
  $A[0, 0] = \varnothing$ ▷ empty hypothesis
  for $t = 0$ **to** $T_x - 1$ **do**
    for $i = 0$ **to** $T_y - 1$ **do**
      $h_{\text{eps}} = A[i, t] + \epsilon$ ▷ expand by blank symbol
      $h_{\text{tok}} = A[i, t] + Y[i]$ ▷ expand by output token
      UpdateTable$(h_{\text{eps}}, A, i, t + 1, (i, t))$
      UpdateTable$(h_{\text{tok}}, A, i + 1, t + 1, (i, t))$
    end for
  end for
  return *reconstruct_path*(A)
end function

function UpdateTable$(h, A, k, l, \text{parent})$
  if exists $A[k, l]$ **and** $(ctc\_score(h) > ctc\_score(A[k, l]))$ **then**
    $A[k, l] = h$
    set_parent$(A[k, l]$, parent$)$
  end if
end function

Algorithm 4 describes the function FindViolation. The beam search proceeds similarly to Algorithm 1 but it keeps tracks of all the derivations in the beam that contain a prefix of the target translation. As soon as there is no such derivation in the beam, the function returns all the hypotheses which stayed in the beam.

Algorithm 4 Find Violation for Beam Search

```
1: function FindViolation(x, y, w)
2:     B_0 ← {∅}                                          ▷ initial beam
3:     for t = 1 to T_x do
4:         B_i ← best_b_hyps(x, B_{i−1})                  ▷ line 3 to line 11 in Algorithm 1
5:         if y_{1:t} ∉ B_i then                          ▷ correct derivation falls off the beam
6:             return (t, B_i)
7:         end if
8:     end for
9:     return (T_x, B_{T_x})                              ▷ update in case the final derivation is incorrect
10: end function
```

The article [5] suggests to perform the update only on the first hypothesis in the beam. We found out that applying the update rule instead to all the hypotheses in the beam leads to faster convergence. At the same time, this approach does not violate the theoretical properties of the algorithm as all the hypotheses that stayed in the beam were incorrect and had higher score than the correct derivation.

Chapter 5

Implementation

This chapter describes specific toolkits and libraries that were used for implementation, experiments, and evaluation of the novel non-autoregressive NMT model.

5.1 Neural Monkey

Neural Monkey[1] is an open-source toolkit for NMT and other sequence-to-sequence tasks [57]. It is actively developed and available under the Berkeley Software Distribution (BSD) license. We use it for prototyping the NMT architectures, training the models, and running the experiments in this thesis.

The core of the Neural Monkey toolkit is the *TensorFlow* machine learning library [58]. The computation in the TensorFlow library is split into two stages – design and compilation of the computational graph, and graph execution on input data. This allows the numerical computations to be performed efficiently. In order to preserve the performance, the core of TensorFlow is implemented in C. The Application Programming Interface (API) is also provided for other programming languages including Python, which is used by Neural Monkey.

In contrast with general frameworks built on top of the TensorFlow library (*tfLearn*[2], *Keras*[3]) which provide abstraction on the level of individual neural network layers, the API of Neural Monkey uses bigger building blocks to enable high-level abstraction on the equation level. The building blocks are the components common in sequence-to-sequence learning, e.g. encoders, decoders or classifiers. In particular, Neural Monkey contains a collection of components implemented according to the recent research papers, including the components relevant for our model as the Transformer encoder and the CTC decoder (described in Sections 2.3.6 and 3.4, respectively).

[1] `https://github.com/ufal/neuralmonkey`
[2] `https://github.com/tflearn/tflearn`
[3] `https://github.com/keras-team/keras/`

The building blocks are parametrized using INI configuration files. The files contain key-value pairs, delimited by the `'='` sign. The key-value pairs are grouped into sections, which are later interpreted as Python objects. The INI configuration files enable to specify e.g. number and sizes of hidden layers of the network without interacting with the actual code. They also provide a comprehensive overview of the model architecture and the parameters, which enables to easily check, modify or re-run the experiments. To give an idea about the structure of a configuration file, one of the configuration files used for the experiments is included in Appendix B.

To run the training, we need to provide the configuration file to the training script:

```
neuralmonkey-train <EXPERIMENT_INI>
```

To run the inference, we also need to provide a configuration file which specifies the location and loading/preprocessing method of the input data:

```
neuralmonkey-run <EXPERIMENT_INI> <DATASETS_INI>
```

Flexibility of the TensorFlow library allows to seamlessly switch between execution on CPU and GPU.

There are other available toolkits for NMT similar to Neural Monkey. OpenNMT [59] is an open-source NMT framework developed by the Harvard NLP group, using PyTorch [4] and TensorFlow. Nematus [60] is a similar framework developed by the Edinburgh NLP group, using Theano[5] (and most recently also TensorFlow). Tensor2Tensor is a more general deep learning library built on top of TensorFlow, which is maintained by Google Brain and which contains a reference implementation of the Transformer model. We use Neural Monkey due to its extensibility, experiment management functionality, up-to-date components, and availability of the implementation of the base model.

5.2 SentencePiece

SentencePiece[6] is a language-independent subword tokenizer and detokenizer [61]. It is an open-source library available under the Apache License. We use SentencePiece for processing the input and output from the model.

SentencePiece implements both BPE and unigram language model algorithms described in Section 2.4. In contrast with other implementations of these algorithms, SentencePiece does not rely on spaces between the words (which may be missing in some languages, e.g. Japanese). Instead it treats the

[4] `https://github.com/pytorch/pytorch`
[5] `https://github.com/Theano/Theano`
[6] `https://github.com/google/sentencepiece`

sentences as a continuous sequence of Unicode characters. The whitespaces are included in the tokens, escaped by the meta-symbol ▁ (U+2581) for clarity. This allows to reverse the tokenization process without any ambiguities. An example of text tokenization is shown in Table 5.1.

tokenization style	raw text	tokenized
language-dependent	Hello world.	[Hello] [world] [.]
	こんにちは世界。	[こんにちは] [世界] [。]
language-independent	Hello▁world.	[Hello] [▁wor] [ld] [.]
	こんにちは世界。	[こんに] [ちは] [世界] [。]

Table 5.1: Standard language-dependent tokenization algorithms rely on interpunction and spaces between the words. The rules for tokenization of Japanese must be implemented manually as there are no spaces between the words. In SentencePiece, the sentences are treated as a continuous sequence of Unicode characters, which makes it language-independent. The tokenization is based on frequencies of sequences of characters, including the whitespaces.

The size of the vocabulary in SentencePiece is determined prior to training, which makes it suitable for NMT models which use fixed vocabulary size. SentencePiece also allows to define rules to handle semantically-equivalent Unicode characters, which can be used e.g. to normalize diacritics in Romanian.

5.3 KenLM

KenLM[7] is an open-source library for training and querying n-gram LMs [51]. It implements advanced data structures to make the queries fast and memory-efficient.

An LM is trained on monolingual data for a particular language. KenLM uses modified Kneser-Ney smoothing (briefly described in Section 4.2.2) for estimating the LM. The training produces a file in ARPA format[8] which contains a list of n-grams, their probabilities and backoff weights.

The maximum n-gram order of the model is specified with a command line parameter `-o`. In the following example, the LM will contain all n-grams of order 1 to 5. The file `text` contains monolingual data, the `text.arpa` is the output file with the LM:

[7] `https://github.com/kpu/kenlm`

[8] a common format supported by language modeling toolkits, developed by Doug Paul at MIT Lincoln Labs for research sponsored by the U.S. Department of Defense Advanced Research Project Agency (ARPA)

```
bin/lmplz -o 5 <text >text.arpa
```

ARPA files can be converted into a binary format for faster loading and querying:

```
bin/build_binary text.arpa text.binary
```

KenLM provides a Python API which we use to incorporate the LM queries into the inference algorithm.

5.4 WarpCTC

CTC loss for training the neural model in Neural Monkey is computed using the implementation provided by the TensorFlow library. At the time of writing, this implementation did not allow to parallelize the computations on GPU, which was unnecessarily slowing down the training process.

Therefore, we use the CTC loss as implemented in WarpCTC[9] instead. WarpCTC is an open-source library available under the Apache license. It provides the *parallel implementation* of the CTC algorithm described in [62]. WarpCTC allows to improve the performance of the training process on GPU while keeping it numerically stable. It also provides Python bindings for TensorFlow which allows to incorporate the library into the rest of the code with only minor modifications.

WarpCTC does not provide the code for CTC decoding. As the decoding cannot be parallelized in any case, we use the corresponding implementation from the TensorFlow library.

5.5 SacreBLEU

Results from the BLEU score evaluation metric described in Section 2.6.1 can be influenced by several parameters, e.g. by setting of the maximum n-gram order or the weight of each n-gram order. If these values differ across the experiments, it is difficult to evaluate and compare the results. This motivated the development of the BLEU score metric with unified parameters.

SacreBLEU[10] is a Python package which provides a way to compute shareable, comparable, and reproducible BLEU scores [63]. It operates on detokenized text, applies its own internal preprocessing, and produces the results as it is adopted by the WMT. SacreBLEU also automatically downloads common test sets for respective language pairs, which ensures that

[9] https://github.com/baidu-research/warp-ctc
[10] https://github.com/mjpost/sacreBLEU

the correct datasets are used in all the experiments. All the BLEU scores in the experiments described in Chapter 6 were calculated using SacreBLEU.

BLEU score from SacreBLEU can be retrieved by the command:

```
cat output.detok | sacrebleu -t wmt14 -l en-de
```

where parameter `-t` specifies the test set and `-l` specifies the language pair.

The output is a single comprehensive string with the result and the parameters used for the evaluation:

```
BLEU+case.mixed+lang.en-cs+numrefs.1+smooth.exp+test.wmt18\
+tok.13a+version.1.2.12 = 14.66 50.2/22.1/11.0/5.8 (BP = 0\
.897 ratio = 0.902 hyp_len = 49309 ref_len = 54652)
```

The first part of the string are the parameters of the BLEU score, the test set, the language pair and the version of SacreBLEU, delimited by the `'+'` sign. After the `'='` sign follows the overall BLEU score to be reported in the results and the scores for particular n-gram orders. The additional information inside the brackets informs about the brevity penalty and the length of sentences compared to the reference.

Chapter 6

Experiments

This chapter evaluates the benefits of the proposed model. It describes the experiments we performed in terms of parallel and monolingual datasets, the model architecture and the parameters for the training. The chapter concludes with a quantitative evaluation of the results and comparison with other related models in terms of decoding speed and translation quality.

6.1 Languages and Datasets

We perform experiments on three language pairs in both directions: *English-German* (en-de), *English-Romanian* (en-ro), and *English-Czech* (en-cs). The models for each language pair (and direction) share the same architecture and training hyperparameters. The difference is in the datasets used for the training of each model and in the weights of the scoring model. We select these particular language pairs in order to make the results comparable with related work.

We preprocess all datasets using SentencePiece (see Section 5.2). We train the SentencePiece models with vocabulary size of 50,000 on parallel datasets for each language pair.

6.1.1 Parallel Datasets

Parallel datasets are used for training the neural models and the SentencePiece models. Each parallel dataset in our experiments is a pair of plaintext files, a file per each language. Each file contains a sentence per line, the sentences on the corresponding lines are translations of each other. There is a single reference translation per sentence, which is a standard in related work.

Table 6.1 describes the parallel datasets used for the training. The datasets are the standard WMT parallel datasets [1] [64–66].

[1] `http://statmt.org/wmt19/translation-task.html`

lang. pair	training data	# sentences	corpus
en-de	WMT15	4.5M	Europarl corpus
en-ro	WMT16	0.6M	Europarl corpus
en-cs	WMT18	57M	movie subtitles

Table 6.1: The parallel datasets used as the training data. The datasets for en-de and en-ro are extracted from the proceedings of European Parliament. The en-cs dataset is extracted from movie subtitles, which manifests in higher volume of the dataset but inferior quality compared to the other datasets.

6.1.2 Monolingual Datasets

Monolingual datasets are used for training the n-gram LM. Each monolingual dataset is a plaintext file with one sentence per line.

Table 6.2 describes the monolingual datasets used for the experiments. The datasets are the standard WMT monolingual datasets[2].

language	data source	# sentences	corpus
English	WMT09	20M	
German	WMT17	20M	News Crawl
Romanian	WMT15	2.2M	corpus
Czech	WMT15	20M	

Table 6.2: The monolingual datasets used for the experiments. For English, German and Czech, we use the first 20M sentences. For Romanian, we use the complete dataset.

6.1.3 Validation and Test Datasets

We use small datasets for validation and testing. The datasets are the standard WMT datasets (see the page with parallel datasets – *Development sets* for validation and *Test sets* for testing). Table 6.3 gives an overview of versions of the datasets we used. Using these particular test sets is necessary to be consistent with the test sets used in the other works.

lang. pair	development	test
en-de	WMT13	WMT15
en-ro	WMT16	WMT16
en-cs	WMT13	WMT18

Table 6.3: The versions of the development and test datasets used for the experiments. Note that even if the version is the same, the datasets for validation and for testing are different.

[2] `http://data.statmt.org/news-crawl/`

6.2 Model Architecture

On top of the principles described in previous chapters, there are several hyperparameters in the model architecture that can influence the performance of the model, the convergence rate and the memory used during the training. We performed preliminary experiments to choose a suitable hyperparameters for the resulting model.

Transformer Variants. There are several variants of the hyperparameter setting for the Transformer architecture described in the original paper [31]. We base the parameters for our model on two Transformer variants, altering the feedforward layer size d_{ff} and the number of heads for the multi-head attention h:

- $d_{\text{ff}} = 2048$, $h = 8$ (Transformer *base*)
- $d_{\text{ff}} = 4096$, $h = 16$ (Transformer *big*).

Shared Embeddings. The Transformer paper [31] suggests sharing the same weight matrix between the embedding layer and the final linear layer before the softmax. This can be interpreted as a dot product between the embeddings and output states, which could stabilize the training and lead to faster convergence. We experiment with both *shared* and *non-shared* variants.

Batch Size. The batch size defines the number of samples which are fed into the neural network before updating its parameters. Using higher batch size requires more memory, but may potentially lead to faster convergence. We experiment with batch sizes of 10 and 20 samples.

Positional Encoding. Positional encoding informs the model about the relative or absolute position of the tokens in the sequence. Following the Transformer architecture, we always add the positional encodings to the encoder input. For the decoder input, we experiment with two variants – *with* (•) and *without* (○) positional encoding added to the input.

We experiment with all the possible combinations of the hyperparameter setting listed above, resulting in 16 different architectures. Table 6.4 summarizes the results of the experiments in terms of BLEU score on the validation dataset.

The models based on *Transformer big* and the models with higher batch size have higher memory requirements and do not exhibit better translation quality. On the contrary, positional encoding and shared embeddings are beneficial in almost all the cases. Taking into account the resulting BLEU score, we select the model architecture based on *Transformer base*, with

Transformer	embeddings	batch size	pos. encoding	BLEU
base	shared	10	•	**19.0**
			○	18.6
		20	•	17.6
			○	17.5
	non-shared	10	•	18.3
			○	17.6
		20	•	17.1
			○	16.9
big	shared	10	•	18.6
			○	18.2
		20	•	18.1
			○	17.8
	non-shared	10	•	15.7
			○	17.5
		20	•	15.9
			○	15.6

Table 6.4: Hyperparameters of the models trained on en-de language pairs and their performance in terms of BLEU score on the validation dataset. The best performing model is the model based on *Transformer base*, with positional encoding, shared embedding and batch size $b = 10$.

positional encoding, shared embedding, and batch size $b = 10$ as the default model architecture for the main experiments.

All these experiments were performed on the en-de language pair. We expect similar behavior for the rest of the language pairs and therefore we use the same model architecture for all the language pairs, as running the experiments separately would be too computationally demanding.

6.3 Training

We train non-autoregressive models for the language pairs en-de, en-ro, and en-cs in both directions. Additionally, we also train corresponding autoregressive models for later comparison. All the models use the same set of hyperparameters as the *Transformer base* model, i.e. model dimension 512, the feedforward layer of dimension 2048 and 8 attention heads. The non-autoregressive models are based on the implementation described in [4] with the splitting factor for the encoder states $k = 3$. The remaining parameters for Neural Monkey are specified in the configuration file in Appendix B.

We train the weights for the structured perceptron separately for each model. We initialize the scoring model with zero weights for all the features, apart from the CTC score for which we set a fixed weight of 1. We split the validation data in halves and use one half as the training set and the

second half as a held-out set. The training set is used for training the weights as described in Section 4.3.2. The held-out set is used for validation – we evaluate the performance of the model on the held-out set during the training and we keep the weights of the model which performed the best.

For both training and evaluation, we use the infrastructure of the *Linguistic Research Cluster*[3] of the Institute of Formal and Applied Linguistics, located at the Faculty of Mathematics and Physics of the Charles University. We use the GPU subsystem for training and CPU subsystem for evaluation. The models are trained on machines with 8 GB of allocated RAM and single GPU *GeForce GTX 1080 Ti* with 11 GB of GPU memory. The evaluation is performed on machines with 32 GB of allocated RAM and *Intel(R) Xeon(R) E5-2630 v4* CPU.

Training each of the non-autoregressive models took approximately two weeks. The models for en-de and en-ro (and vice versa) were trained for 10 epochs.[4]

6.4 Results

We evaluate two aspects of the NMT models – translation quality and translation speed (i.e. latency). In order to evaluate the benefits of all the suggested improvements, we perform an ablation study for the features of the scoring model. We also add several examples illustrating the model behavior with different parameter settings.

Table 6.5 summarizes the BLEU score of the models. The top part contains the results of non-autoregressive models described in Chapter 3. The middle part contains the results of the autoregressive models used as a baseline. The bottom part contains the results of our improved model. Some results are missing as authors did not include the results in their published work.

We observe that the autoregressive models usually outperform the non-autoregressive models in terms of BLEU score. The difference is the smallest for en-ro and ro-en, which can be caused by the fact that monolingual datasets bring greater benefits for low-resource language pairs. We also observe that the beam search greatly improves the translation quality over the end-to-end model with CTC.

Table 6.6 summarizes the results of the models in terms of decoding time per sentence. The low latency is the main benefit of the non-autoregressive models. Although we do not achieve the same translation quality as the autoregressive models with the same decoding time, we can achieve better

[3] `https://wiki.ufal.ms.mff.cuni.cz/grid`

[4] *Training epoch* = using all input data for training the model. The en-cs and cs-en models were trained for approximately same amount of time, which resulted in less epochs due to larger amount of training data.

Method	WMT15		WMT16		WMT18	
	en-de	de-en	en-ro	ro-en	en-cs	cs-en
End-to-End w/ CTC [4]	19.71	21.64	18.45	25.48	13.92	14.87
Latent Fertility [44]	-	-	27.29	29.06	-	-
Iterative Refining [46]	12.65	14.48	24.45	23.73	-	-
AR Transformer, greedy	26.39	28.56	19.91	27.33	16.00	22.72
AR Transformer, beam 5	26.99	29.39	20.81	27.99	17.08	23.54
Ours, beam 1	20.81	22.68	18.49	26.27	14.29	15.05
Ours, beam 5	23.29	25.96	18.56	28.68	14.64	16.99
Ours, beam 10	23.99	26.19	18.68	28.99	14.66	17.51
Ours, beam 20	24.01	26.59	18.77	29.23	14.75	17.73

Table 6.5: Results of the models in terms of BLEU score. Authors of [44] used the WMT14 en-de dataset, which is not comparable with the rest of the results. We present the results of [44] and [46] using a single pass through the model.

speed with only a minor decrease in quality. Furthermore, we observe that we can control the speed/quality trade-off by either lowering or increasing the beam size.

Method	Decoding time per sentence [ms]
End-to-End w/ CTC [4]	314
AR Transformer, greedy	1637
AR Transformer, beam 5	4093
Ours, beam 1	347
Ours, beam 5	435
Ours, beam 10	597
Ours, beam 20	1382

Table 6.6: Results of the models in terms of decoding time per sentence. The time is computed as the time to decode a single sentence, averaged over the whole test set. Note that we do not include other non-autoregressive models as they were tested with different frameworks and infrastructure, and the results are generally not comparable.

Figure 6.1 plots the time required to translate a sentence with respect to its length. We see that the greedy decoding of the end-to-end model with CTC is nearly constant in the length of the source sentence. The beam search decoding of our model is more time-consuming, however, it is still a lot faster than the autoregressive model. The difference holds especially for longer sentences, where the autoregressive model gets prohibitively slow.

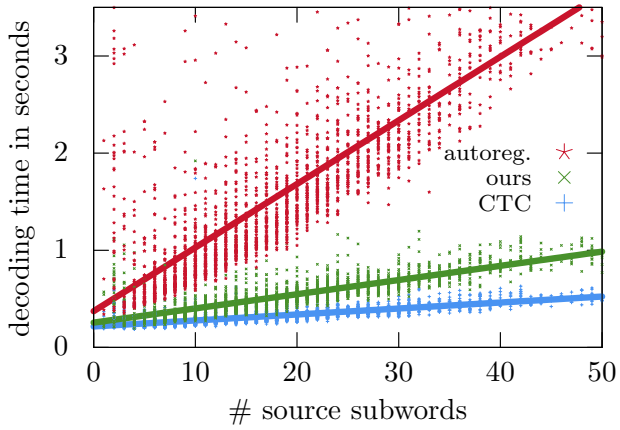

Figure 6.1: Comparison of the CPU decoding time of the autoregressive model (red), the proposed method with beam size of 10 (green), and the end-to-end non-autoregressive model with CTC (blue).

For illustration, Table 6.7 the shows the difference between the translations with different beam sizes on a single example from the cs-en model. The larger beam size generally leads to more precise inference.

cs (ref)	**Colin odporuje tím, že to není pravda a obviňuje Paula z nesprávného nakládání s některými nemovitostmi.**
$b = 1$	Colin contradicts it' not true and and accusing Paul of mising of certain real estate.
$b = 5$	Colin contradicts it's not true and and accusing Paul of mising of certain real estate.
$b = 10$	Colin contradicts it's not true and accusing Paul of mising of certain real estate.
$b = 20$	Colin contradicts that it's not true and accusing Paul of mising of certain real estate.
en (ref)	*Colin argues this is not true, but accuses Paul of incompetently managing some properties.*

Table 6.7: Translations of the cs-en model with different beam sizes b.

Furthermore, we perform an ablation study to show the benefits of particular features in the scoring model. Table 6.8 shows how the features contribute to the BLEU score. We can see that combining the features is beneficial and that the improvement is substantial with larger beam sizes. The feature weights were trained separately for each beam size.

Table 6.9 illustrates the benefits of the features on manually selected sentences. Note that without a LM, there are word compounds like *Aggressivitätivität* or *sindschilder* which do not make sense in German. On the other hand, LM can be harmful in case of proper names which are not common. In these cases, other features can help the model to get closer to the correct translation.

Beam Size	1	5	10	20
$c + l + r + t$	20.81	23.29	23.99	24.01
$c + l + r$	20.58	23.22	23.56	23.35
$c + l$	19.31	21.94	22.59	22.70
c	19.71	20.09	20.17	20.19

Table 6.8: BLEU scores for en-de translation model for different beam sizes and feature sets: CTC score (c), language model (l), ratio of the blank symbols (r), and the number of trailing blank symbols (t).

a)	**On account of their innate aggressiveness, songs of that sort were no longer played on the console.**
b)	Aufgrund ihrergeboren Aggressivitätivität wurden Lieder dieser Art nicht mehr auf der Konsole gespielt.
c)	Aufgrund ihrer Aggressivität wurden Lieder dieser Art nicht mehr auf der Konsole gespielt.
d)	Aufgrund ihrergeborenen Aggressivität wurden Lieder dieser Art nicht mehr auf der Konsole gespielt.
e)	Aufgrund ihrer angeborenen Aggressivität wurden Lieder dieser Art nicht mehr auf der Konsole gespielt.
f)	*Aufgrund ihrer ur eigenen Aggressivität wurden Songs dieser Art nicht mehr auf der Konsole gespielt.*
a)	**Ailinn didn't understand.**
b)	A hat nicht.
c)	Das war nicht.
d)	Die hat nicht verstanden.
e)	Aili hat nicht verstanden.
f)	*Ailinn verstand das nicht.*
a)	**Further trails are signposted, which lead up towards Hochrhön and offer an extensive hike.**
b)	Weitere Wege sindschilder, die nach Hochrhön und eine ausgedehnte Wanderung.
c)	Weitere Wege sind, die in Hochrhön und eine ausgedehnte Wanderung.
d)	Weitere Wege sindschilder, die in Hochrhön und eine ausgedehnte Wanderung.
e)	Weitere Wege sind ausgeschilder, die in Hochrhön und eine ausgedehnte Wanderung.
f)	*Weitere Wege sind ausgeschildert, die Richtung Hochrhön hinaufsteigen und zu einer ausgedehnten Wanderung einladen.*

Table 6.9: Manually selected sentences illustrating the benefits of the features on the en-de model. Following the notation in Table 6.8, the sentences are: a) English reference, b) model output (c), c) model output (c+l), d) model output (c+l+r), e) model output (c+l+r+t), f) German reference.

Chapter 7

Conclusion

In this thesis, we introduced an improvement of a non-autoregressive NMT model. We started by summarizing the history of the field of MT and the recent developments in the subfield of NMT, including its drawbacks. We followed by an introduction of non-autoregressive NMT, which is trying to tackle some of the drawbacks. In particular, we explained the end-to-end model with CTC on which we build upon in this thesis. We proposed an extension of this model which incorporates a LM and other features into the scoring model in order to improve the fluency of translations. We implemented the model using the Neural Monkey framework and we evaluated the benefits of the approach in a series of experiments.

The experiments show that the main benefit of the proposed approach is the opportunity to balance the tradeoff between translation quality and translation speed. The autoregressive models are still superior in translation quality in most of the language pairs, even though by a narrow margin. In contrast, the non-autoregressive models are very fast, but often lack in translation quality. Our approach keeps the asymptotically constant decoding speed of non-autoregressive models (in the number of decoder runs), but uses a beam search with an extended scoring model to improve the translation quality. By altering the beam size, we can adjust the speed and the quality ratio to achieve acceptable results in both of the domains.

We defined specific features which can improve the performance of the model. The main feature is the LM trained on monolingual data which improves the fluency of the translations. Other features help to tackle the shortness of the translated sentences. We presented an ablation study which shows the benefit of each of the features. We also performed a cursory manual evaluation to confirm our intuition about the role of each of the features.

In the future, we will aim to add our approach directly into the Neural Monkey framework. To improve our approach, we can experiment with employing other features into the scoring model or adjusting the parameters of the existing features, which may work differently on other language pairs. In order to achieve even better results in terms of latency, we can implement

the decoding process and other critical parts of the code in C++. It would also be interesting to inspect the influence of monolingual datasets with the process of *back-translation* which could be used to enhance the results of the base CTC model and the autoregressive baselines.

Bibliography

[1] Google. Twenty years of building for everyone. `https://blog.google/inside-google/company-announcements/twenty-years-building-everyone/`, accessed: 2019-03-23.

[2] Bahdanau, D.; Cho, K.; et al. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015.

[3] Luong, M.-T.; Pham, H.; et al. Effective approaches to attention-based neural machine translation. *EMNLP*, 2015.

[4] Libovický, J.; Helcl, J. End-to-End Non-Autoregressive Neural Machine Translation with Connectionist Temporal Classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 3016–3021. Available from: `http://www.aclweb.org/anthology/D18-1336`

[5] Huang, L.; Fayong, S.; et al. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics, 2012, pp. 142–151.

[6] Hutchins, W. J. *Machine translation: past, present, future*. Springer, 1986.

[7] Ide, N.; Véronis, J. Introduction to the special issue on word sense disambiguation: the state of the art. *Computational linguistics*, volume 24, no. 1, 1998: pp. 2–40.

[8] Chéragui, M. A. Theoretical overview of machine translation. *Proceedings ICWIT*, 2012: p. 160.

[9] Hutchins, J. Machine translation: A concise history. *Computer aided translation: Theory and practice*, volume 13, 2007: pp. 29–70.

[10] Kalchbrenner, N.; Blunsom, P. Recurrent Continuous Translation Models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2013, pp. 1700–1709. Available from: `http://aclweb.org/anthology/D13-1176`

[11] Sutskever, I.; Vinyals, O.; et al. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[12] Cho, K.; van Merrienboer, B.; et al. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, Association for Computational Linguistics, 2014, pp. 103–111, doi:10.3115/v1/W14-4012. Available from: `http://aclweb.org/anthology/W14-4012`

[13] Hardmeier, C. On statistical machine translation and translation theory. In *Proceedings of the Second Workshop on Discourse in Machine Translation*, 2015, pp. 168–172.

[14] Brown, P. F.; Cocke, J.; et al. A statistical approach to machine translation. *Computational linguistics*, volume 16, no. 2, 1990.

[15] Och, F. J. *Statistical machine translation: from single-word models to alignment templates*. Dissertation thesis, Bibliothek der RWTH Aachen, 2002.

[16] Cristina Espana i Bonet. Statistical Machine Translation: A practical tutorial. `http://www.lsi.upc.edu/~cristinae/CV/docs/tutorialSMTprint.pdf`, accessed: 2019-05-20.

[17] Baum, L. E.; Petrie, T.; et al. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, volume 41, no. 1, 1970: pp. 164–171.

[18] Krizhevsky, A.; Sutskever, I.; et al. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[19] Hinton, G.; Deng, L.; et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, volume 29, 2012.

[20] Koehn, P. Neural Machine Translation. *CoRR*, volume abs/1709.07809, 2017, `1709.07809`. Available from: `http://arxiv.org/abs/1709.07809`

[21] Johnson, M.; Schuster, M.; et al. Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. *Transactions of the Association for Computational Linguistics*, volume 5, no. 1, 2017: pp. 339–351. Available from: `http://www.aclweb.org/anthology/Q17-1024`

[22] Facebook AI. Transitioning entirely to neural machine translation. https://code.fb.com/ml-applications/transitioning-entirely-to-neural-machine-translation/, accessed: 2019-04-18.

[23] Forcada, M. L. Making sense of neural machine translation. *Translation spaces*, volume 6, no. 2, 2017: pp. 291–309.

[24] Mikolov, T.; Chen, K.; et al. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[25] Mikolov, T.; Sutskever, I.; et al. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[26] Bengio, Y.; Ducharme, R.; et al. A neural probabilistic language model. *Journal of machine learning research*, volume 3, no. Feb, 2003: pp. 1137–1155.

[27] Olah, Christopher. Understanding LSTM Networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/, accessed: 2019-05-21.

[28] Graves, A. Long short-term memory. In *Supervised sequence labelling with recurrent neural networks*, Springer, 2012, pp. 37–45.

[29] Cho, K.; van Merrienboer, B.; et al. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734, doi:10.3115/v1/D14-1179. Available from: https://www.aclweb.org/anthology/D14-1179

[30] Helcl, Jindrich and Libovicky, Jindrich. Attentive Sequence-to-Sequence Learning. https://ufal.mff.cuni.cz/~helcl/courses/npfl116/slides/03-sequence-to-sequence.pdf, accessed: 2019-05-21.

[31] Vaswani, A.; Shazeer, N.; et al. Attention is all you need. In *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[32] Google Brain. Tensor2Tensor Intro. https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb, accessed: 2019-05-21.

[33] He, K.; Zhang, X.; et al. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[34] Ba, J.; Kiros, R.; et al. Layer Normalization. *CoRR*, volume abs/1607.06450, 2016.

[35] Sennrich, R.; Haddow, B.; et al. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 2016, pp. 1715–1725.

[36] Gage, P. A new algorithm for data compression. *The C Users Journal*, volume 12, no. 2, 1994: pp. 23–38.

[37] Kudo, T. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 66–75.

[38] Tillmann, C.; Ney, H. Word reordering and a dynamic programming beam search algorithm for statistical machine translation. *Computational linguistics*, volume 29, no. 1, 2003: pp. 97–133.

[39] Freitag, M.; Al-Onaizan, Y. Beam search strategies for neural machine translation. *arXiv preprint arXiv:1702.01806*, 2017.

[40] Papineni, K.; Roukos, S.; et al. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 2002. Available from: `http://aclweb.org/anthology/P02-1040`

[41] Isozaki, H.; Hirao, T.; et al. Automatic evaluation of translation quality for distant language pairs. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2010, pp. 944–952.

[42] Snover, M.; Dorr, B.; et al. A study of translation edit rate with targeted human annotation. In *Proceedings of association for machine translation in the Americas*, volume 200, 2006.

[43] Lewis, T. G.; El-Rewini, H. *Introduction to parallel computing*. Prentice-Hall, Inc., 1992.

[44] Gu, J.; Bradbury, J.; et al. Non-Autoregressive Neural Machine Translation. *CoRR*, volume abs/1711.02281, 2017, `1711.02281`. Available from: `http://arxiv.org/abs/1711.02281`

[45] Kim, Y.; Rush, A. M. Sequence-Level Knowledge Distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Austin, Texas: Association for Computational Linguistics, 2016, pp. 1317–1327, doi:10.18653/v1/D16-1139. Available from: `https://www.aclweb.org/anthology/D16-1139`

[46] Lee, J.; Mansimov, E.; et al. Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 1173–1182.

[47] Graves, A.; Fernández, S.; et al. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, ACM, 2006, pp. 369–376.

[48] Hannun, A. Sequence Modeling with CTC. *Distill*, 2017, doi:10.23915/distill.00008, https://distill.pub/2017/ctc.

[49] Çaglar Gülçehre; Firat, O.; et al. On Using Monolingual Corpora in Neural Machine Translation. *CoRR*, volume abs/1503.03535, 2015.

[50] Graves, A.; Jaitly, N. Towards End-To-End Speech Recognition with Recurrent Neural Networks. In *ICML*, 2014.

[51] Heafield, K. KenLM: Faster and smaller language model queries. In *Proceedings of the sixth workshop on statistical machine translation*, Association for Computational Linguistics, 2011, pp. 187–197.

[52] Ney, H.; Essen, U.; et al. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, volume 8, no. 1, 1994: pp. 1–38.

[53] Bergstra, J.; Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, volume 13, no. Feb, 2012: pp. 281–305.

[54] Lafferty, J. D.; McCallum, A.; et al. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, ISBN 1-55860-778-1, pp. 282–289. Available from: `http://dl.acm.org/citation.cfm?id=645530.655813`

[55] Collins, M. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, Association for Computational Linguistics, 2002, pp. 1–8.

[56] Tsochantaridis, I.; Joachims, T.; et al. Large margin methods for structured and interdependent output variables. *Journal of machine learning research*, volume 6, no. Sep, 2005: pp. 1453–1484.

[57] Helcl, J.; Libovický, J. Neural Monkey: An Open-source Tool for Sequence Learning. *The Prague Bulletin of Mathematical Linguistics*, , no. 107, 2017: pp. 5–17, ISSN 0032-6585, doi:10.1515/pralin-2017-0001. Available from: `http://ufal.mff.cuni.cz/pbml/107/art-helcl-libovicky.pdf`

[58] Abadi, M.; Agarwal, A.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. 2015. Available from: `http://download.tensorflow.org/paper/whitepaper2015.pdf`

[59] Klein, G.; Kim, Y.; et al. OpenNMT: Open-Source Toolkit for Neural Machine Translation. *Proceedings of ACL 2017, System Demonstrations*, 2017: pp. 67–72.

[60] Sennrich, R.; Firat, O.; et al. Nematus: a Toolkit for Neural Machine Translation. In *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, 2017, pp. 65–68.

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

[62] Amodei, D.; Ananthanarayanan, S.; et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, 2016, pp. 173–182.

[63] Post, M. A Call for Clarity in Reporting BLEU Scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, Belgium, Brussels: Association for Computational Linguistics, 2018, pp. 186–191. Available from: `https://www.aclweb.org/anthology/W18-6319`

[64] Bojar, O.; Chatterjee, R.; et al. Findings of the 2015 Workshop on Statistical Machine Translation. In *WMT*, 2015.

[65] Bojar, O.; Chatterjee, R.; et al. Findings of the 2016 Conference on Machine Translation. In *WMT*, 2016.

[66] Bojar, O.; Federmann, C.; et al. Findings of the 2018 Conference on Machine Translation (WMT18). In *WMT*, 2018.

Appendix A

Acronyms

AI Artificial Intelligence

API Application Programming Interface

ARPA Advanced Research Project Agency

BLEU Bilingual Evaluation Understudy

BPE Byte Pair Encoding

BSD Berkeley Software Distribution

CPU Central Processing Unit

CTC Connectionist Temporal Classification

DNN Deep Neural Network

GPU Graphical Processing Unit

LM Language Model

MIT Massachusetts Institute of Technology

MT Machine Translation

NLP Natural Language Processing

NMT Neural Machine Translation

RAM Random Access Memory

RNN Recurrent Neural Network

SMT Statistical Machine Translation

TER Translation Edit Rate

WMT Workshop on Machine Translation

Appendix B

Configuration File

A Neural Monkey configuration file used for training the en-de model.

```
[vars]
prefix="/lnet/spec/work/people/kasner/neuralmonkey-ctc-decoder"
exp_prefix="{prefix}/experiments"
runners_batch_size=40
vocab_size=50000
trainer_batches_per_update=10
src="en"
tgt="de"
suffix="-kombo-shared-b{trainer_batches_per_update}-v{vocab_size}"
use_pos=False
langpair="{src}{tgt}"
data_prefix="{prefix}/data/{langpair}"
ende_src_train="wmt_data.{src}.{vocab_size}.spm"
ende_tgt_train="wmt_data.{tgt}.{vocab_size}.spm"
ende_src_val="validation.{src}.{vocab_size}.spm"
ende_tgt_val="validation.{tgt}.{vocab_size}.spm"
src_train_name=$ende_src_train
tgt_train_name=$ende_tgt_train
src_val_name=$ende_src_val
tgt_val_name=$ende_tgt_val

[main]
name="EN -> DE, SAN > split states > CTC"
tf_manager=<tf_manager>
output="{exp_prefix}/{langpair}-san_ctc{suffix}"
epochs=10
train_dataset=<train_data>
val_dataset=<val_data>
trainer=<trainer>
runners=[<runner>]
postprocess=None
evaluation=[("target", evaluators.bleu.BLEU)]
logging_period="10m"
validation_period="2h"
runners_batch_size=$runners_batch_size
```

```
random_seed=1234
overwrite_output_dir=False
batching_scheme=<batch_scheme>

[batch_scheme]
class=dataset.BatchingScheme
batch_size=800
token_level_batching=True

[tf_manager]
class=tf_manager.TensorFlowManager
num_threads=12
num_sessions=1
save_n_best=5

[train_data]
class=dataset.load
series=["source", "target"]
data=["{data_prefix}/{src_train_name}","{data_prefix}/{
    tgt_train_name}"]

[val_data]
class=dataset.load
series=["source", "target"]
data=["{data_prefix}/{src_val_name}","{data_prefix}/{tgt_val_name
    }"]

[vocabulary]
class=vocabulary.from_wordlist
path="/lnet/spec/work/people/kasner/spm/models/{vocab_size}/sp.{
    langpair}.{vocab_size}.vocab"
contains_frequencies=True
contains_header=False

[input_sequence]
class=model.sequence.EmbeddedSequence
vocabulary=<vocabulary>
data_id="source"
embedding_size=512
scale_embeddings_by_depth=True
max_length=64

[encoder]
class=encoders.transformer.TransformerEncoder
input_sequence=<input_sequence>
ff_hidden_size=2048
depth=6
n_heads=8
dropout_keep_prob=0.9
attention_dropout_keep_prob=0.9

[state_split]
```

```
class=model.sequence_split.SequenceSplitter
parent=<encoder>
projection_size=1536
factor=3

[second_encoder]
class=encoders.transformer.TransformerEncoder
input_sequence=<state_split>
ff_hidden_size=2048
depth=6
n_heads=8
dropout_keep_prob=0.9
attention_dropout_keep_prob=0.9
input_for_cross_attention=<encoder>
n_cross_att_heads=8
use_positional_encoding=$use_pos

[decoder]
class=decoders.CTCDecoder
name="decoder"
max_length=64
encoder=<second_encoder>
data_id="target"
vocabulary=<vocabulary>
input_sequence=<input_sequence>

[obj]
class=trainers.cross_entropy_trainer.CostObjective
decoder=<decoder>

[trainer]
class=trainers.delayed_update_trainer.DelayedUpdateTrainer
clip_norm=1.0
batches_per_update=$trainer_batches_per_update
objectives=[<obj>]
optimizer=<adam>

[adam]
class=tf.contrib.opt.LazyAdamOptimizer
beta1=0.9
beta2=0.997
epsilon=1.0e-9
learning_rate=1.0e-4

[runner]
class=runners.plain_runner.PlainRunner
decoder=<decoder>
output_series="target"
```

Appendix C

CD Contents

The attached CD contains source codes used for this thesis. Note that the input datasets and the language models are not included on the medium because of their excessive size. The scripts for submitting the jobs may contain some specific options for the job scheduling engine of the Linguistic Research Cluster.

```
/
├── src ..... project source codes
│   ├── scripts.tar.gz ..... scripts
│   ├── kenlm.tar.gz ..... KenLM
│   ├── neuralmonkey.tar.gz ..... Neural Monkey
│   ├── sentencepiece.tar.gz ..... SentencePiece
│   └── warpctc.tar.gz ..... WarpCTC
├── thesis ..... thesis files
│   ├── src ..... thesis LaTeX source code
│   │   └── img ..... thesis figures
│   └── thesis.pdf ..... thesis in PDF format
└── README.txt ..... CD content description
```