FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

# DOCTORAL THESIS

Jindřich Helcl

# Non-Autoregressive Neural Machine Translation

Institute of Formal and Applied Linguistics

Supervisor: prof. RNDr. Jan Hajič, Dr.

Study Program: Computer Science
Specialization: Computational Linguistics

Prague 2021

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

Prague, November 15, 2021 Jindřich Helcl

**Title:** Non-Autoregressive Neural Machine Translation

**Author:** Jindřich Helcl

**Department:** Institute of Formal and Applied Linguistics

**Supervisor:** prof. RNDr. Jan Hajič, Dr.,
Institute of Formal and Applied Linguistics

**Abstract:**

In recent years, a number of mehtods for improving the decoding speed of neural machine translation systems have emerged. One of the approaches that proposes fundamental changes to the model architecture are non-autoregressive models. In standard autoregressive models, the output token distributions are conditioned on the previously decoded outputs. The conditional dependence allows the model to keep track of the state of the decoding process, which improves the fluency of the output. On the other hand, it requires the neural network computation to be run sequentially, and thus it cannot be parallelized. Non-autoregressive models impose conditional independence on the output distributions, which means that the decoding process is parallelizable and hence the decoding speed improves. A major drawback of this approach is lower translation quality compared to the autoregressive models. The goal of the non-autoregressive translation research is to find methods that improve the translation quality, while retaining high decoding speed. In this thesis, we explore the research progress so far and identify flaws in the generally accepted evaluation methodology. We experiement with non-autoregressive models trained with connectionist temporal classification. We find that even though our models achieve state-of-the-art performance on the standard WMT 14 benchmark, there is a large room for improvement when we compare non-autoregressive methods to highly optimized autoregressive models.

**Keywords:** machine translation, deep learning, natural language processing

**Název práce:** Neautoregresivní neuronový strojový překlad

**Autor:** Jindřich Helcl

**Katedra:** Ústav formální a aplikované lingvistiky

**Vedoucí práce:** prof. RNDr. Jan Hajič, Dr.,
Ústav formální a aplikované lingvistiky

**Abstrakt:**

V poslední době nabídl výzkum strojového překladu nové metody pro zrychlení generování. Jedním z navrhovaných metod je takzvaný neautoregresivní neuronový strojový překlad. V klasických autoregresivních překladových systémech jsou výstupní pravděpodobnostní rozdělení modelována podmíněně na předchozích výstupech. Tato závislost umožňuje modelům sledovat stav překládání a obvykle vede ke generování velmi plynulých textů. Autoregresivní postup je však ze své podstaty sekvenční a nelze jej paralelizovat. Neautoregresivní systémy modelují pravděpodobnosti jednotlivých cílových slov jako navzájem podmíněně nezávislé, což znamená, že dekódování lze paralelizovat snadno. Nevýhodou je ovšem nízká kvalita překladu ve srovnání s modely autoregresivními. Cíl výzkumu neautoregresivních metod strojového překladu je zlepšit kvalitu překladu a zároveň uchovat vysokou rychlost dekódování. Naše práce předkládá rešerši publikovaných metod a poukazuje na některé nedostatky plynoucí z obecně přijímané evaluační metodologie. Popisujeme experimenty s neautoregresivními modely trénovaných pomocí takzvané „connectionist temporal classification“. Z našich výsledků plyne, že i když dosahujeme nejlepších výsledků mezi neautoregresivními modely na datech z WMT z roku 2014, při porovnání s nejnovějšími optimalizovanými autoregresivními systémy tyto modely pořád zaostávají.

**Klíčová slova:** strojový překlad, hluboké učení, zpracování přirozených jazyků

## Acknowledgements

I would like to thank my supervisor, prof. RNDr. Jan Hajič, Dr. for his support throughout my study.

I thank all my colleagues and friends at ÚFAL for the wonderful time I had studying at Charles University. With special thanks to Jindra Libovický, with whom I spent countless hours of research and fun working on papers that made this thesis possible. I would also like to thank Lexi Birch and Barry Haddow at the University of Edinburgh for the valuable feedback on my research during the past year and a half.

I am especially grateful to my wife Gábina and my family for their love and endless support.

This work was supported by the Czech Science Foundation, grant No. 19-26934X (NEUREM 3), Charles University grant No. 52315/2014, and by the SVV grant No. 260 575. This work has been using language resources and tools developed and/or stored and/or distributed by the LINDAT/CLARIN project of the Ministry of Education, Youth and Sports of the Czech Republic (project LM2015071). This work was conducted within the EU projects Gourmet, under grant agreement No 825299, and Bergamot, under grant agreement No 825303. It was also supported by the UK EPSRC grant EP/S001271/1 (MT-Stretch). It was performed using computing resources in CSD3 using funding from the UK EPSRC (capital grant EP/P020259/1).

# Contents

# 1 Introduction

Machine translation (MT) has always been the holy grail of computational linguistics. The first attempts to use algorithms for translation date back to before the field of computational linguistics was even established (Dupont, 2017). By the 1950s, researchers began to develop automatic translation systems on computers (Dostert, 1955). During the following decades, the use of rule-based translation backed up by a linguistic theory has been the main paradigm. Even in the early days, MT systems were used in a number of applications, such as the translation of technical manuals and similar texts with a simple structure and low language variety (Slocum, 1985).

In the late 1980s and early 1990s, technological progress enabled computers to work with larger textual data, and the first statistical methods for MT were conceived (Nagao, 1984; Brown et al., 1990). Instead of having roots in linguistics, statistical MT is based on information theory (Shannon, 1948). This paradigm shift made it possible to use the same methods for different language pairs without additional costs to design the rules for translation systems. As the computational power of computers and the size of the available data and the computational power of the grew bigger throughout the 1990s and 2000s, statistical methods became the standard approach to MT (Koehn, 2009). In those days, automatic MT started to attract wide interest from the public with the first general-purpose translation systems available on the Internet.

In parallel with MT research, the field of artificial intelligence (AI) has written its own history. From the first neurology-inspired algorithms in the 1950s and the formulation of the Turing test (Turing, 1950), through the so-called AI winter periods and the development of artificial neural networks and the backpropagation algorithm, culminating during the last decade in the advent of deep learning (Goodfellow et al., 2016). Not unlike MT, the research advances in AI owe their existence to faster and more powerful computers.

In the past decade, the fields of AI and MT fused together, resulting in a new paradigm of *neural machine translation* (NMT; Sutskever et al., 2014; Bahdanau et al., 2014). Soon after NMT was established as the new state-of-the-art approach to translation, high-performing systems became available online for the general public, and the whole field of MT started growing rapidly. Whereas in the early stages, MT development was focused on a few selected languages, mostly translating in or out of English, nowadays the research is slowly expanding towards low-resource languages or to different areas such as multimodal or multilingual translation (Haddow et al., 2021; Libovický and Helcl, 2017; Aharoni et al., 2019). One of the emerging research subfields motivated by making NMT models more accessible is *non-autoregressive* NMT, which is the main topic of this thesis.

This thesis is structured as follows. In Chapter 2, we introduce the field of NMT. We show how models for NMT process textual data and how they deal with the large number of different words they need to understand. We introduce the concept of encoder-decoder architectures, which are the backbone of nearly every NMT system there is, and their two types, RNN-based models and the Transformer. We give an overview of how to train these models, including some practical aspects of training, such as what methods do we use for data cleaning, or what training algorithms do we use, and how to configure them. We briefly talk about methods for decoding applied once the model has been trained. We conclude the chapter by discussing the evaluation, which tells us how good or how bad our translation models are.

Chapter 3 walks us through the research on non-autoregressive NMT (NAT), explaining the principal differences between autoregressive and non-autoregressive models, what issues arise when using the latter, and how we can solve them. Since the whole domain of NAT research is fairly new, we provide a comprehensive survey of the literature. We describe efforts to keep the most important feature of NAT models – the high decoding speed – intact by introducing constraints on the training process, or other efforts that partially sacrifice the higher decoding speed for better

translation quality by iterative refinement. At the end of the chapter, we identify the weaknesses prevalent in a large part of the literature on this topic. One of the main problems lies in the evaluation methodology adopted by the NAT research community.

We present a method for training non-autoregressive models for NMT with connectionist temporal classification (CTC) in Chapter 4. We first describe the CTC algorithm for computing the loss between the unaligned label and the prediction sequences. Having such a loss function relaxes the requirement for target length prediction, which is one of the obstacles in NAT. We show the changes we need to apply to the model architecture so that we can use the CTC loss, and we report on our experiments and their results, both in terms of speed and translation quality. Our analysis shows that NAT models suffer from reduced fluency of the translations, which is in line with the conclusions of other NAT studies. We briefly mention our attempt to improve translation fluency using language model rescoring.

In Chapter 5, we document our recent efforts to address the flaws in the evaluation methodology discussed in Chapter 3 with our CTC-based model from Chapter 4. We compare our results to methods published in the context of NAT, as well as methods that focus directly on improving the efficiency of translation systems. We find that despite achieving high performance among NAT models, using efficient autoregressive translation models leads to better results, both in terms of speed and translation quality. We argue that, for better estimation of the benefits of non-autoregressive methods, future research should take into account the methods employed in production-ready systems. We conclude the thesis in Chapter 6.

# 2

# Neural Machine Translation

Neural machine translation (NMT) is the current state-of-the-art approach to machine translation (MT). As the name suggests, NMT systems use artificial neural networks to model the translation. The specific types, or *architectures*, of neural networks used in an NMT system vary across history, applications, or domains, but all share a few common traits.

Neural networks are a mathematical model that works with real numbers. The inputs and outputs to a neural network are real-valued vectors, the training objective is a differentiable function, and the space of the parameters needs to have a continuous gradient with respect to the loss function. Specific to neural networks is the computation of the gradient by the *backpropagation* algorithm and the wide variety of optimization methods.

In Section 2.1, we explain how to adapt neural networks to the discrete nature of the textual data. Perhaps the most common way is to assign a real-valued vector to each word in the vocabulary. These representations are usually trained along with the whole network end to end.

In NLP tasks like sentence classification or language modeling, there is only a single sequence on the input. In MT, we have a source sentence on the input and a target sentence on the output. To reflect this property, most NMT architectures have two components: an *encoder* and a *decoder*. The encoder processes the input sentence and creates a hidden representation. The decoder then accesses this hidden representation and generates (or scores) the output sentence. When both components of the model are trained end to end, this framework is sometimes referred to as *sequence-to-sequence learning*. We explore the encoder-decoder architectures more closely in Section 2.2.

Textual data are sequential, and the lengths of the sequences are variable. Therefore, the use of feed-forward neural networks is not suitable. Sequences can be processed by a number of network architectures, namely convolutional neural networks (CNNs), recurrent neural networks (RNNs), or self-attentive networks (SANs). All of those can be adapted for text processing, and we will describe the latter two in more detail in Sections 2.2.1 and 2.2.2, respectively.

NMT models are trained using large parallel corpora, by minimizing the negative log-likelihood of the data. Section 2.3 formalizes the supervised training framework for translation models and summarizes the training methodology, such as data processing, model initialization, and optimization.

During training, sequence-to-sequence decoders usually run in the *teacher forcing* mode – for a given source sentence and its translation, the neural network is trained to maximize the likelihood of the $i$-th target word on the $i$-th position, conditioned on the $i-1$ previous words of the reference sentence. During decoding, when this information is not available, the model is conditioned on previously decoded outputs instead. More details on this phenomenon and other practical considerations and techniques such as beam search or ensembling are presented in Section 2.4.

MT is evaluated by automatic metrics such as BLEU and by human evaluation. This chapter concludes with a brief overview of the evaluation of NMT systems (Section 2.5).

## 2.1 Processing Text with Neural Networks

There are a number of problems that arise when we want to use neural networks for text processing. First and foremost, neural networks are a mathematical model that deals with real numbers, whereas language is discrete and has a complex structure. Thus, we need to find ways to express textual data numerically. Second, since language units (such as words, sentences, or documents) come in various lengths, we need to use neural network models designed for processing sequential data.

In this section, we begin with the representation of words. The most straightforward and also the most common way to represent words in neural networks is to have a list (a *vocabulary*) of words $\mathcal{V}$ of size $N$, where each word has a corresponding integer index $j$ pointing at it. Each word $w$ at index $j$ in this list can be represented by a *one-hot* vector $x \in \{0,1\}^N$ where

$$x_i = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

Using one-hot representation helps us convert discrete words to numbers, but it does not capture anything about the underlying linguistic structure. For this purpose, *learned distributed feature vectors*, or *word embeddings* are used (Bengio et al., 2003; Collobert and Weston, 2007). The idea is to associate a real-valued vector with each word, so that representations of words with similar roles in language are closer together in the assigned vector space. Mikolov et al. (2013) train these word embeddings using the *skip-gram* and *continuous bag-of-words* objectives and find that the resulting embeddings have interesting properties. For example, arithmetic operations on embeddings can express word analogy. In some natural language processing (NLP) tasks, pre-trained embeddings can be used to greatly improve the model performance on the end task. In contemporary NMT research, the most prevalent approach is to train the word embeddings end to end along with the translation model.

We provide the model with the distributed representation of words using an embedding layer as follows. For a given one-hot vector $x$ and a parameter matrix $E$ called the *embedding matrix*, we retrieve a single row that corresponds to the vocabulary index of $x$ by multiplication $Ex$. The embedding dimension is usually much lower than the size of the vocabulary (in NMT, the embedding dimension in state-of-the-art models is around 1,024).

A major drawback of having a fixed vocabulary is the fact that there will be unseen words in the data. The following section lists methods that solve this problem.

### 2.1.1 Open Vocabulary Problem

A well-known characteristic of language is that it follows Zipf's law (Zipf, 1949). As a result, in a large enough sample of text, there is a huge amount of unique words or words occurring with low frequency. Such words constitute a substantial part of the data and cannot be ignored. Another issue is that our training datasets do not contain all words that can occur in the language. These out-of-vocabulary (OOV) words usually also make up significant portions of the validation and test data.

The problem of rare and unseen words is also referred to as the *open vocabulary* problem. Although some work in this direction exists (Jean et al., 2015), in neural networks for NLP, it is difficult to use very large vocabularies. The main reason is that the embedding layer and the output projection layer would become too large for the computation to be efficient.

The most common solution of the open vocabulary problem is to use smaller units than words (Sennrich et al., 2016b). These *subword* units can be constructed in a smart way, such that frequent words are left intact, whereas rare words are composed of more common strings of letters. In the following, we describe three of

the methods for subword segmentation. It is worth mentioning that the research on character-level methods is still ongoing (Chung et al., 2016; Lee et al., 2017; Gao et al., 2020), but as of the writing of this thesis, this approach did not yet outperform the state-of-the-art subword segmentation.

**BPE.** Byte-pair encoding (BPE; Sennrich et al., 2016b) is an approach that tackles the open vocabulary problem by splitting words into subword units. The idea is to devise a vocabulary of a predefined size such that any word can be composed using the units from the vocabulary (with the exception of words containing unknown characters). An additional requirement is for the vocabulary to contain whole words that are frequent in the data, so only rare words are split into more subwords. The BPE method works with a vocabulary of merges, which needs to be created by the BPE training algorithm before it can be applied on the data.

The BPE training algorithm proceeds in iterations as follows. At first, all the tokens in the data are split into characters, and the vocabulary is initialized with the list of the characters. To allow for later reconstruction of the original text, a special end-of-word character is appended to each word. Each iteration of the algorithm has three steps. First, the algorithm computes the counts of pairs of consecutive symbols (bigrams) in the data (respecting the word boundaries), and selects the most frequent pair. Second, the selected bigram is merged and added to the vocabulary. Third, the new vocabulary is used to segment the data again. These three steps are repeated until a predefined number of merges are performed. Note that in practice, the algorithm can be implemented to work only with the frequency list of tokens, and not with the whole training corpus, without loss of generality.

Once the BPE merge list is created, it can be applied on the data by first splitting the words in the data to characters, and then applying the learned merges on the split words. Reconstruction of the original text can be done by removing the spaces and then replacing the end-of-word characters with new spaces.

**Wordpiece.** The wordpiece algorithm (Schuster and Nakajima, 2012; Wu et al., 2016) shares the ideas of the BPE algorithm – whole words are split into sequences of characters, which are then merged according to a list of merges. Unlike the BPE segmentation, the wordpiece learning algorithm does not take the most frequent bigram, but instead the pair with the largest pointwise mutual information.

**SentencePiece.** More recently, Kudo and Richardson (2018) implemented SentencePiece, a toolkit for subword segmentation using either BPE, or a unigram language model (Kudo, 2018). It supports a number of features, such as sampling and regularization by introducing noise on the source side. As opposed to BPEs and word-

pieces, SentencePiece does not require prior tokenization of the input text, and unlike other methods its pre-tokenization allows to fully reconstruct the original string. The Marian toolkit (Junczys-Dowmunt et al., 2018) used in our experiments described in Chapter 5 has built-in support for SentencePiece tokenization and segmentation.

## 2.2 Encoder-Decoder Framework

The contemporary NMT models share a common framework where each model is composed of two parts – an *encoder* and a *decoder*. The encoder reads the input sentence and processes it to generate an intermediate hidden representation. The decoder then uses this intermediate hidden representation to produce the probability distributions over the output tokens.

The early NMT models based on RNNs use the final encoder state as the intermediate representation (Sutskever et al., 2014):

$$h_j = \mathrm{RNN}_{\mathrm{enc}}(x_j, h_{j-1}), \quad j \in \{0, 1, \ldots, T_x\} \tag{2.2}$$

$$s_0 = h_{T_x} \tag{2.3}$$

where $\mathbf{x}$ is the input sentence and $h_0$ is the initial hidden state, usually set to $\mathbf{0}$. $T_x$ denotes the length of the input sentence. Note that the input words are represented as their embeddings. Sutskever et al. (2014) do not use subword segmentation and instead reserve a special OOV token for unseen words.

The decoder is initialized with the state $s_0$ and runs the second RNN:

$$s_i = \mathrm{RNN}_{\mathrm{dec}}(y_{i-1}, s_{i-1}) \tag{2.4}$$

where $y_{i-1}$ is the preceding word in the reference output sentence (during training), with $y_0$ being a special symbol that expresses the start of a sequence (denoted <s>).

While recurrent neural networks are not used as the underlying model in most of the current research anymore, the encoder-decoder framework is independent of the actual neural network structure and the concept is still used as the main approach for designing sequence-to-sequence models.

In this section, we introduce the two most notable encoder-decoder architectures. The first is based on RNNs and became the first neural architecture to outperform statistical MT models. The second architecture, called *Transformer*, is based on self-attentive networks and is the best performing architecture for many NLP tasks today.

### 2.2.1 Recurrent Neural Networks

The invention of RNNs (Elman, 1990) allowed processing of sequential data by neural networks. RNNs process the sequence one item at a time, chaining consecutive steps with recurrence connections.

In the early stages of the RNN development, a single hidden layer of the network was altered to take into account the output of itself from the previous time step:

$$h_t = \tanh(Wx_t + Uh_{t-1} + b_h) \tag{2.5}$$

where $W \in \mathbb{R}^{m\times n}$, $U \in \mathbb{R}^{n\times n}$, $b_h \in \mathbb{R}^n$ are trainable parameters, $x_t \in \mathbb{R}^m$ is the RNN input, and $h_{t-1} \in \mathbb{R}^n$ is the previous hidden state.

However, this version of RNNs suffers from the *vanishing gradient problem*. During backpropagation, the gradients are multiplied by the derivative of the hyperbolic tangent, which is always less than or equal to one. In long sequences, the learning signal over distant parts of the sequence therefore has almost no effect on the training. Due to this fact, the network manifests poor performance in handling long-range dependencies.

There have been several approaches to combat the vanishing gradient problem. The most prevailing types of RNN architectures in NMT are Long Short-Term Memory (LSTM) networks and Gated Recurrent Unit (GRU) networks.

**LSTM.** Long Short-Term Memory networks (Hochreiter and Schmidhuber, 1997; Gers et al., 2000) introduce gating mechanisms and a concept of *information highway*, which ensures that only linear operations are applied on the states in the recurrent chain. A gating mechanism is an operation that computes a number between 0 and 1, which we refer to as a *gate value*. The output of the operation is the input multiplied by the gate value.

Given a current time step $t$, input $x_t \in \mathbb{R}^m$, and previous hidden states $h_{t-1}, C_{t-1} \in \mathbb{R}^n$, LSTM networks proceed as follows:

$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right) \tag{2.6}$$
$$i_t = \sigma\left(W_i x_t + U_i h_{t-1} + b_i\right) \tag{2.7}$$
$$o_t = \sigma\left(W_o x_t + U_o h_{t-1} + b_o\right) \tag{2.8}$$
$$\tilde{C}_t = \tanh\left(W_c x_t + U_c h_{t-1} + b_c\right) \tag{2.9}$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{2.10}$$
$$h_t = o_t \odot \tanh C_t \tag{2.11}$$

where $W_f, W_i, W_o, W_c \in \mathbb{R}^{n \times m}$, $U_f, U_i, U_o, U_c \in \mathbb{R}^{n \times n}$, $b_f, b_i, b_o, b_c \in \mathbb{R}^n$ are trainable parameters, $\sigma$ is the logistic function, and $\odot$ represents element-wise multiplication. The states $h_t$ and $C_t$ are also called public and private hidden states, respectively. The intermediate value $\tilde{C}_t$ is called the candidate state.

The three gates in the LSTM network are called the *forget gate*, the *input gate*, and the *output gate*. The forget gate (Eq. 2.6) controls how much of the information flows from the previous private hidden state $C_{t-1}$ to the current private state (Eq. 2.10). The input gate (Eq. 2.7) controls the amount of information received from the candidate state. Finally, the output gate (Eq. 2.8) decides which portion of the currently computed private hidden state is transferred to the current public hidden state (Eq. 2.8). Note that the values of the gates are computed in the same way, but with different parameters.

The original recurrence relation from Equation 2.5 is expressed by Equation 2.9, where the new candidate state is computed. The transfer of information from the previous private state and the current candidate state is done in Equation 2.10. Note that with respect to the states from the preceding steps, the previous state is only multiplied by a constant. This constitutes the information highway that allows propagating the gradients over long distances in the sequence.

**GRU.** Gated Recurrent Unit networks (Cho et al., 2014) are an alternative to LSTMs. Instead of four sets of parameter matrices, GRUs need only three, while maintaining the theoretical strength. Unlike LSTMs, GRUs use only a single hidden state in the recurrence relations.

Given the time step $t$, the input $x_t \in \mathbb{R}^m$, and the previous hidden state $h_{t-1} \in \mathbb{R}^n$, the GRU step is defined as follows:

$$r_t = \sigma\left(W_r x_t + U_r h_{t-1} + b_r\right) \tag{2.12}$$

$$z_t = \sigma\left(W_z x_t + U_z h_{t-1} + b_z\right) \tag{2.13}$$

$$\tilde{h}_t = \tanh\left(W x_t + U\left(r_t \odot h_{t-1}\right) + b\right) \tag{2.14}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{2.15}$$

where $W, W_z, W_r \in \mathbb{R}^{n \times m}$, $U, U_z, U_r \in \mathbb{R}^{n \times n}$, and $b, b_z, b_r \in \mathbb{R}^n$ are trainable parameters.

The two gates in GRU networks are called the *reset gate* and the *update gate*. The reset gate (Eq. 2.12) determines how much information from the previous state is preserved and is applied in the recurrence relation for computing the candidate (Eq. 2.14). The update gate (Eq. 2.13) controls the merging of the previous state with the candidate state in Equation 2.15. Note that the information highway concept is expressed by this equation.

**Bidirectional RNNs.** There is a bidirectional variant of RNNs that can be applied to any of the flavors of RNNs discussed above. When the whole input sequence is known, we can apply an RNN in both directions separately and then concatenate the states from the corresponding positions:

$$\overrightarrow{h}_t = \overrightarrow{\mathrm{RNN}}(x_{t-1}, \overrightarrow{h}_{t-1}) \tag{2.16}$$

$$\overleftarrow{h}_t = \overleftarrow{\mathrm{RNN}}(x_{t+1}, \overleftarrow{h}_{t+1}) \tag{2.17}$$

$$h_t = \begin{bmatrix} \overrightarrow{h}_t \\ \overleftarrow{h}_t \end{bmatrix} \tag{2.18}$$

**Attention Mechanism.** Another important concept introduced as part of RNN-based NMT models is the *attention mechanism* (Bahdanau et al., 2014; Luong et al., 2015).

The problem with the encoder-decoder framework as described in the previous section is that there is an information bottleneck between the encoder and the decoder. All the information from the source sentence needs to be compressed into a single hidden state vector $s_0$.

The attention mechanism enables the decoder to access the information stored in the encoder hidden states rather than relying only on the value of the initial state. Formally, based on a current decoder step $s_i$, the attention mechanism computes a local *context vector* as a weighted average of the encoder states:

$$e_{ij} = v_a^\top \tanh(W_a s_{i-1} + U_a h_j + b_a) + b', \tag{2.19}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}, \tag{2.20}$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \tag{2.21}$$

In the first step, the *attention energies* are computed for every encoder hidden state using a single-hidden-layer feed-forward network parameterized by $W_a \in \mathbb{R}^{n' \times n}$, $U_a \in \mathbb{R}^{n' \times 2n}$ (in a bidirectional network), $b_a \in \mathbb{R}^{n'}$, $v_a \in \mathbb{R}^{n'}$, and $b' \in \mathbb{R}$ where $n'$ is the dimension of the hidden layer. Next, the attention energies are normalized using the softmax function into the *attention distribution*. Finally, the context vector is computed as a weighted sum of the encoder states $h_0, \ldots, h_{T_x}$, using the attention distribution values as weights.

The concept of attention can be viewed as a soft-lookup function over an associative memory. Given a *query*, which is the decoder state in a given time step, we use a similarity metric over a set of *keys*. We then normalize the similarities and use them as weights in the weighted sum of the *values* associated with the corresponding keys. In this case, the sets of attention keys and values are equal to the encoder hidden states $h_j$. The similarity metric is defined in Equation 2.19.

**Deep RNNs.** Some NMT architectures based on RNNs use multiple recurrent layers (Miceli Barone et al., 2017; Wu et al., 2016). In deep RNNs, each layer is a recurrent network. The inputs of the second and each layer onwards are the outputs of the preceding layer. The output of the entire deep RNN is the output of the last layer.

In deep RNNs, there are a few settings to consider. To improve the convergence speed of the deep models and to stabilize the RNN hidden state dynamics across layers, *layer normalization* (Ba et al., 2016) is often employed. For layer $l$ and its output states $\mathbf{h}^l \in \mathbb{R}^{T_x \times n}$, we compute:

$$\mu^l = \frac{1}{nT_x} \sum_{i=1}^{T_x} \sum_{j=1}^{n} h_{ij}^l \tag{2.22}$$

$$\sigma^l = \sqrt{\frac{1}{nT_x} \sum_{i=1}^{T_x} \sum_{j=1}^{n} (h_{ij}^l - \mu^l)^2} \tag{2.23}$$

$$\bar{\mathbf{h}}^l = \frac{\mathbf{h}^l - \mu^l}{\sigma^l} \odot \gamma^l + \beta^l \tag{2.24}$$

where $\mu^l$ and $\sigma^l$ are the mean and standard deviation of the output state values, $\gamma^l, \beta^l \in \mathbb{R}^{T_x \times n}$ are learnable *gain* and *bias* parameters, and $\odot$ denotes element-wise multiplication. Normalized states $\bar{\mathbf{h}}^l$ are then used as input to the $(l+1)$-th layer of the network instead.

Another common technique in multilayer RNNs is to tie the vector spaces of the RNN states on different layers with residual connections. In such networks, the input to the $(l+1)$-th layer is the sum of the output of the $l$-th layer with its input.

### 2.2.2 Transformer Model

One of the disadvantages of RNN-based models is the sequential nature of the recurrence relation. A few models have been proposed to remove the recurrence, thus allowing for simultaneous computation across time steps, at least at the training time. Most notably, these models include convolutional architectures (Gehring et al., 2017), and the current state-of-the-art architecture, the *Transformer* model (Vaswani et al., 2017).

Instead of recurrence relations, the Transformer model uses *self-attention* layers, stacked into a deep network. The states in each layer can be computed independently on each other, which leads to training speed improvements. We follow with a detailed description of the Transformer model components, illustrated in Figure 2.1.

**Positional Encoding.** Since self attention is a commutative operation, the ordering of the input tokens needs to be modeled explicitly. The standard technique is to use sinusoidal encodings, as proposed in the original paper. Positional encoding is a vector of the same dimension as the word embedding, which is computed using the position of the word on the input. The $2i$-th and $(2i+1)$-th elements of the positional encoding of word on position $j$ is computed as follows:

$$\begin{aligned} E_{\text{pos}}(j, 2i) &= \sin(j/10000^{2i/d}) \\ E_{\text{pos}}(j, 2i+1) &= \cos(j/10000^{2i/d}) \end{aligned} \tag{2.25}$$

where $d$ is the *model dimension*. This number is equal to the dimension of the word embeddings. Before the network processes an input word $w_i$, the positional encodings are added to the word embeddings:

$$E(w_i) = E(w) + E_{\text{pos}}(i) \tag{2.26}$$

**Self-Attention.** The key component of the Transformer model is self-attention. As the name suggests, self-attention is run with the same set of states used as queries, keys, and values. The scaled dot product is used as the similarity metric, so the attention itself does not use any trainable parameters. Formally, self-attention is defined as a function:

$$\mathcal{A}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V \tag{2.27}$$

where the values $Q, K, V \in \mathbb{R}^{T \times d}$ ($T$ being the sequence length) are computed from the same input, usually using a linear projection.

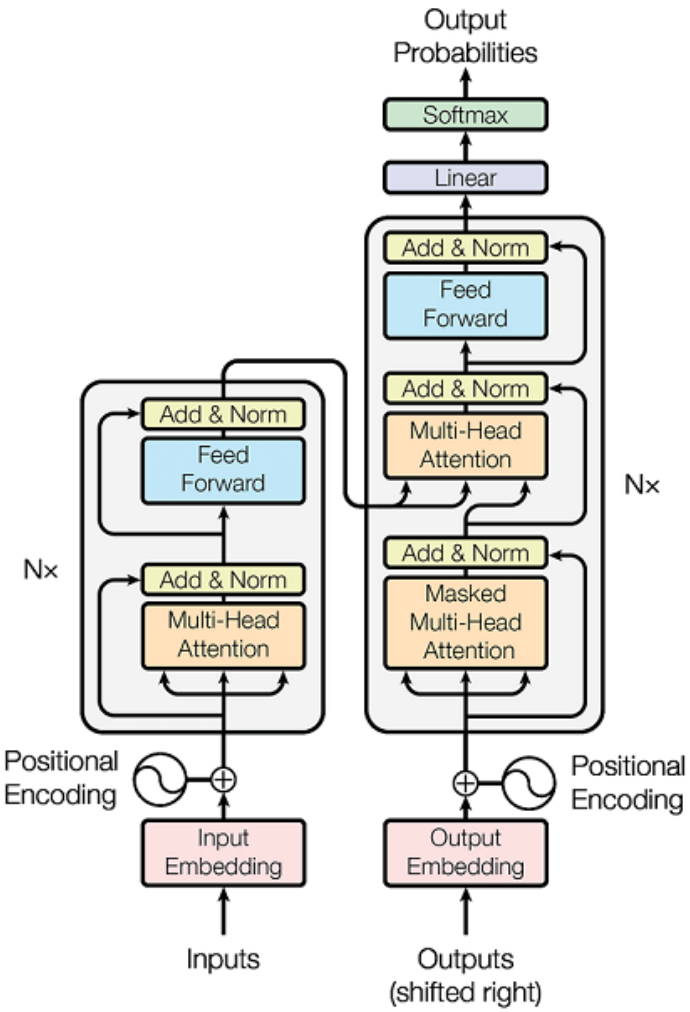



Figure 2.1: The overall schema of the Transformer model. The inputs are summed with positional encodings. The encoder processes the input in $N$ stack layers. Each encoder layer consists of self-attention and feed-forward sub-layer. Layers are interconnected with residual connections and layer normalization is applied on the output of each layer. The decoder stack uses three sub-layers, where the additional cross-attention layer attends to the encoder output. We show the original image from Vaswani et al. (2017), Figure 1.

Self-attention sub-layers are used both in the encoder and the decoder. Because the decoder is autoregressive, the self-attention module must be limited to attend only to the preceding states (so that the model does not "glance into the future"). This is implemented by applying a triangular mask (also *causal mask*) on the attention query and key matrix, hence the name *masked self-attetnion*.

**Multi-Head Attention.** Instead of running a single self-attention per layer, the model structure is enriched by splitting the self-attention into multiple *heads*. First, each state is projected into $h$ triples of queries, keys, and values. Then, self-attention is computed on each triple. Finally, the self-attention outputs are mixed together into a single output sequence:

$$\mathcal{A}^h(Q, K, V) = \sum_{i=1}^{h} C_i W_i^O \tag{2.28}$$

where $W_i^O \in \mathbb{R}^{d_h \times d}$ is a parameter matrix used to project the outputs from the individual attention heads into a single output, and

$$C_i = \mathcal{A}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.29}$$

where $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d \times d_h}$ are trainable matrices that project the states to inputs for the $i$-th attention head (defined in Equation 2.27). In self-attention, $Q = K = V \in \mathbb{R}^{T \times d}$ is the set of $T$ states of the current layer.

**Cross-Attention.** Each Transformer encoder layer consists of a self-attention and a feed-forward sub-layer. The decoder inserts an *encoder-decoder attention* (or *cross-attention*) layer. The cross-attention layer functions exactly like the self-attention layer, but the queries $Q$ are the decoder states (i.e. the output of the preceding self-attention), and the keys and values $K = V$ correspond to the encoder output states.

**Feed-Forward Layer.** Each Transformer layer consists of self-attention, optionally a cross-attention, followed by a feed-forward network. This feed-forward network uses the same parameters for all positions along the state sequence. The network takes a state $x$ and feeds it through a single hidden layer with a ReLU activation:

$$\mathcal{F}(x) = \max(0, W_1^F x + b_1^F) W_2^F + b_2^F \tag{2.30}$$

where $W_1^F \in \mathbb{R}^{d \times d_f}$, $b_1 \in \mathbb{R}^{d_f}$, $W_2^F \in \mathbb{R}^{d_f \times d}$, and $b_2^F \in \mathbb{R}^d$ are the weight and bias parameters of the feed-forward network $\mathcal{F}$, and $d_f$ is the dimension of the hidden state.

**Residual Connections and Layer Normalization.** As a post-processing step, the output of each sub-layer is connected with the output of the previous sub-layer with residual connections. Similarly to the RNN-based deep architectures, layer normalization is used in combination, which ties all states in the sub-layer stack into a shared vector space.

**Output Projection.** The states of the last decoder layer are projected to the size of the vocabulary and are interpreted as unnormalized probability distributions over the output tokens. When a probability distribution is needed (during training or beam search decoding), the scores are normalized using the softmax function:

$$p(y_t|y_{<t}, x, \theta) = \text{softmax}(W^S s_t + b^S) \tag{2.31}$$

where $W^S \in \mathbb{R}^{d \times |\mathcal{V}|}$, $b \in \mathbb{R}^{|\mathcal{V}|}$ are the output projection parameters, $s_t$ is the $t$-th state in the last decoder layer, and $\theta$ is the set of model parameters. Note that the probability of the $t$-th output token $y_t$ is conditioned on the preceding tokens $y_{<t}$ because of the causal mask in the decoder self-attention. In the Transformer model, it is possible to tie the values of the output projection matrix with the transposed embedding matrix. This reduces the number of model parameters and can have advantages over separate input and output projection matrices when using a shared source–target vocabulary.

To compute the probability distribution of the whole target sentence $y$ given the input sentence $x$, the distributions from all time steps are multiplied together:

$$p(y|x) = \prod_{t=1}^{T_y} p(y_t|y_{<t}, x, \theta) \tag{2.32}$$

where $T_y \in \mathbb{R}$ is the length of the target sentence, $y_{<t}$ are the previously decoded words.

**Model Hyperparameters.** The parameters that control the model size are the model dimension $d$, the number of attention heads $h$, the dimension of the feed-forward hidden layer $d_f$, and the number of Transformer layers in the encoder and decoder stack. Note that the dimension of keys and values in a single attention head $d_h$ is commonly set to be $d/h$, but can be customized as well. As a regularization, dropout (Srivastava et al., 2014) is applied with rate $P_d$ to the output of each sub-layer before normalization.

The authors of the architecture propose two presets for these parameters, named *base* and *big*. Table 2.1 shows the hyperparameter values for these two settings.

| | # of layers | $d$ | $h$ | $d_h$ | $d_f$ | $P_d$ |
|---|---|---|---|---|---|---|
| Transformer base | 6 | 512 | 8 | 64 | 2,048 | 0.1 |
| Transformer big | 6 | 1,024 | 16 | 64 | 4,096 | 0.3 |

Table 2.1: The hyperparameter values for Transformer base and big variants.

## 2.3 Training

Training of NMT models is usually done under *supervised* conditions, using a dataset of parallel sentences. The size of the available data varies greatly with different language pairs. Although there is no formal definition, a language pair for which less than a million sentence pairs are available is usually referred to as *low-resource*. For many languages, even a million sentences is a very large number compared to what is actually available. Data quality is also a factor. For example, when the only available data is crawled from the web, data cleaning can filter out a major part of the corpus. However, in this thesis, we focus on language pairs where the data size is not a major issue.

Translation models are trained by minimizing the loss function, usually expressed by cross entropy between the output distribution and the one-hot distribution that assigns zero probabilities to all but the correct target word. Assuming that $y_i$ is the correct target word at position $i$, $p_{\text{ref}}$ is the one-hot distribution, and $p$ is the distribution predicted by the model, we have the following:

$$\begin{aligned} H(p_{\text{ref}}, p) &= -\sum_{w \in \mathcal{V}} p_{\text{ref}}(w) \log p(w) = \\ &= -\log p(y_i) \end{aligned} \tag{2.33}$$

where $\mathcal{V}$ is the vocabulary. Note that in autoregressive models, the output distributions $p$ and $p_{\text{ref}}$ are conditioned on the preceding target words.

Given model parameters $\theta$, the word-level cross entropy is summed across the sentence pairs in the data $D$ to obtain the negative log-likelihood of the dataset $J(\theta)$:

$$\begin{aligned} J(\theta) &= -\sum_{(x,y) \in D} \sum_{i=1}^{T_y} \log p(y_i | x, y_{<i}, \theta) \\ &= -\sum_{(x,y) \in D} \log p(y | x, \theta) \end{aligned} \tag{2.34}$$

where $y_{<i}$ denote the target prefix and $T_y$ is the length of the target sentence $y$. The probability of a sentence can be reformulated using the chain rule.

The goal of training is to find parameters $\theta^*$ that minimize the cost function defined above:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, J(\theta) \tag{2.35}$$

**Teacher Forcing.** Note that in the equations above, the probability distributions are conditioned on the reference target prefix, rather than the model output. This technique is known as *teacher forcing* and is essential for the convergence of training. When the model is exposed to its own outputs from the start of the training, it is very likely that it will fail to converge.

Teacher forcing, however, brings along a problem called *exposure bias*: the model is never exposed to its own errors, which makes it less robust against them.

Methods have been proposed to address this issue, including a curriculum learning approach which gradually replaces teacher forcing with the model predictions (Bengio et al., 2015). Other methods focus on sequence-level training or beam search optimization methods, mostly based on reinforcement learning (Williams, 1992; Wiseman and Rush, 2016; Daumé et al., 2009; Ranzato et al., 2016). However, none of these methods has been widely adopted, and current models are believed to be capable of recovering from their errors, despite being explicitly trained to do so.

### 2.3.1 Training Methodology

In this section, we go through common techniques for successful training of NMT models. These include data cleaning and augmentation methods, optimization settings, and hardware considerations.

**Data Cleaning.** With a few exceptions for high-resource languages, the training data is usually acquired from the Web. Depending on the data source and the extraction technique, there is a level of noise present in the data. In many cases, it is necessary to consider data cleaning before training any models.

The basic data cleaning techniques are rule-based and include language identification, and filtering sentence pairs with odd characters or length ratios. Deduplication may be considered, but might be harmful when it removes short sentences which appear commonly in the given language. For example, if the English sentences "Thank you" and "Thank you xxx" both appear in the data as translations of the Czech "Děkuji", they might get the same probabilities when deduplication is used. Without deduplication, the frequency of the correct translation would be much higher in the training data, but other problems may arise, especially in data crawled from the Web, where there are many repeated snippets from page headers or footers which do not occur naturally so often.

An advanced data cleaning technique is dual conditional cross-entropy filtering (Junczys-Dowmunt, 2018). Using two translation models trained on clean data in opposite directions, each sentence pair is scored according to the cross entropies assigned by the translation models to the sentence pair. When the cross entropies differ or when both cross entropy scores are high, the score is low. When the cross entropies are similar and are low, the score is high. After scoring, low-scoring sentence pairs are removed from the data using an empirically set threshold. Formally, each sentence pair $(x, y)$ is scored with the opposite models $A$ and $B$ according to the following formula:

$$s = |H_A(y|x) - H_B(x|y)| + \frac{1}{2}(H_A(y|x) + H_B x|y) \tag{2.36}$$

where $H_A$ and $H_B$ are the cross entropies of the two models, normalized over words:

$$H_A(y|x) = -\frac{1}{T_y} \log p(y|x) \tag{2.37}$$

and similarly for $H_B(x|y)$.

**Data Augmentation.** The size of the training data is a key factor in modeling performance. In rare cases of high-resource languages, there is enough parallel data available to train a decent translation model. But even for these languages, data augmentation methods, namely *backtranslation*, are used to create bigger data and therefore to improve the model performance.

Backtranslation is a simple technique for incorporating target-side monolingual data in a translation model (Bojar and Tamchyna, 2011; Sennrich et al., 2016a). For a given translation direction, we first translate the monolingual data from the target language to the source language using a previously trained model. Then, we mix the synthetic source-language data along with the authentic target-language data into our training corpus. When the parallel-to-monolingual data size ratio is not balanced, one can consider oversampling the smaller part of the corpus to mitigate this. Recently, Caswell et al. (2019) showed that labeling the synthetic data with a special token improves the translation quality. As pointed out by Marie et al. (2020), this helps because the model is less prone to overfitting to this type of data.

*Knowledge distillation* (Kim and Rush, 2016) is another data augmentation technique. Unlike backtranslation, knowledge distillation is used mainly to improve the efficiency of the model, both in terms of memory and speed. Also, we use synthetic target-language data. In the simplest setting, we use a well-trained and large *teacher* model to translate its own training data. Then, we train a smaller (and therefore

faster and smaller) *student* model on the data generated by the teacher model. This technique shows that for a small sacrifice in translation quality, we can get interesting improvements in terms of speed, which is useful for deploying the models in a limited environment, such as a mobile device.

**Batching.** In Equation 2.34, the loss function $J(\theta)$ is defined as a sum of sequence-level losses over dataset $D$. We use the gradient descent algorithm to find the parameters $\theta^*$ with the minimum loss value. The algorithm proceeds in iterations, updating the parameters using current gradients. However, the exact computation of the gradient is inefficient because it requires a pass through the whole dataset. Therefore, we estimate the gradient on a small sample of the training data, called a *mini batch*, or simply a batch. This method is called *stochastic gradient descent (SGD)*.[1]

The size of a mini batch is an important parameter, and its value can have a large impact on the training convergence. Bigger batches provide better gradient estimates, but require more memory, which is a concern when using a GPU.

Because the data gets split into batches that serve as random samples of the data, a suitable data shuffling strategy should be considered. If we regard the dataset as a collection of sentences (as opposed to documents, for example), shuffling the sentences in the whole dataset before training (or before each *epoch*, i.e. a pass over the training data) is in most cases sufficient.

Another aspect of batching is how efficiently we use the memory allocated for the batch with respect to the maximum sentence length. The batches are represented as matrices, where each row corresponds to a sentence and each column corresponds to a token position in the sentence. Sentences shorter than the longest sentence in the batch are padded to the maximum length. If the overall amount of padding used in a batch is large, the optimization step takes longer. Therefore, we try to use batches of sentences with similar lengths. A possible implementation is to load a number of batches beforehand, sort the sentences by length, and then re-batch the sorted list of sentences. In the literature, this technique is called *bucketing*.

In some cases, it is possible to delay the parameter update for a number of batches. For example, large models tend to fit into a GPU memory only with a small batch size. In such cases, we might choose to aggregate the gradients over multiple batches to simulate a larger batch size. A similar strategy can be applied when the training uses multiple GPUs.

[1] Some sources use the term *mini-batch gradient descent*, to distinguish from methods that perform the update for each example.

**Curriculum Training.** The ordering of the batches in training – the *curriculum* – influences both the training process (see bucketing above) and the actual performance of the trained model. Kocmi and Bojar (2017) experiment with various curriculum scenarios such as starting training on the shorter sentences and gradually including more complex ones, which has been shown to somewhat improve the final model. Popel et al. (2020) propose switching between larger blocks of authentic and synthetic data during training.

**Optimization.** Because the gradients are estimated, heuristics can be used to improve convergence. There are different sets of heuristics commonly used together, called *optimizers*. In a standard SGD, the update rule for every mini-batch $b = (x_i, y_i)_{i=0}^{n}$ follows:

$$\theta \leftarrow \theta - \alpha \cdot \nabla_\theta J(\theta) \tag{2.38}$$

where $\nabla_\theta J$ is the gradient of the loss function and $\alpha$ is the *learning rate*. The learning rate is a hyperparameter that controls the size of the update steps.

There are more advanced optimization techniques than plain SGD. Adagrad, Adadelta, RMSProp and Adam are among the most widely used optimizers, working with higher order derivatives or moving averages of the gradients (Duchi et al., 2011; Zeiler, 2012; Tieleman and Hinton, 2012; Kingma and Ba, 2014).

The learning rate $\alpha$ is often set to gradually decrease during training, a technique called *learning rate decay*. Specific to the Transformer model, Vaswani et al. (2017) define the *Noam* learning rate decay scheme as follows:

$$\alpha = d^{-0.5} \cdot \min(i^{-0.5}, i \cdot w^{-1.5}) \tag{2.39}$$

where $i$ is the number of training steps that have elapsed, $d$ is the dimension of the Transformer model, and $w$ is a hyperparameter controlling the number of *warmup* steps. This learning rate scheme increases the learning rate linearly during the first $w$ training steps, then proceeds to gradually decrease $\alpha$ proportionally to the inverse square root of the number of the current update. Figure 2.2 illustrates how the model dimension and the number of warmup steps influence the learning rate value.

**Parameter Initialization.** Most often, the model parameters are initialized randomly. Uniform or normal distributions are common choices. A more involved initialization approach is to take into account the size of incoming and outgoing connections in the network to specify the bounds (for uniform distribution) or the standard deviation (for normal distribution) (Glorot and Bengio, 2010). Model ensembling can be achieved by training a number of models with the same configuration but different random initialization.

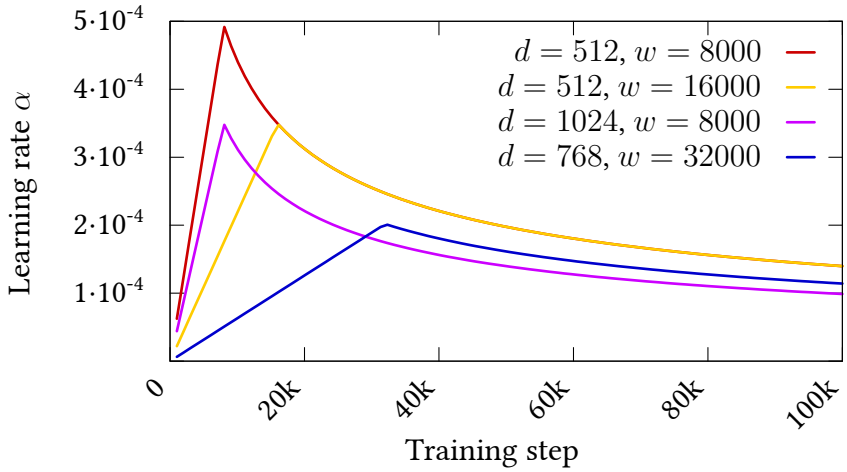



Figure 2.2: The learning rate in the Noam learning rate scheme over the course of training for different values of model dimension $d$ and warmup steps $w$.

Another technique that uses parameter initialization is transfer learning (Zoph et al., 2016). In this scenario, we have two similar tasks, such as machine translation between different language pairs, where the primary task is considerably more difficult than the auxiliary task (for example, due to different amounts of training data available). Transfer learning assumes that the models for both tasks share some amount of common knowledge. First, a *parent* model is trained on the auxiliary task (e.g. MT between a high-resource language pair). Then, the *child* model is initialized with the trained parameters of the parent model. Finally, the child model is further trained on the primary task (e.g. a low-resource MT).

The idea of transfer learning is one of the main concepts in applications of pre-trained language models such as BERT (Devlin et al., 2019), or multilingual pre-trained models applied to MT, such as mBART (Liu et al., 2020).

**Early Stopping.** To avoid overfitting, the model should be regularly tested on a validation dataset during the course of the training. Validation data should not overlap with training or test data. Once the model stops improving on a chosen validation metric, the training process is interrupted. This method is called *early stopping*. Usually, the validation metric is either the cross-entropy loss or the target evaluation metric, such as BLEU. An alternative early stopping method is to save the parameters of the best performing model on validation data.

**Hardware.** Since the advent of deep learning methods for MT, the neural network architectures have started to grow. However, this breakthrough could not have happened if there was no hardware to allow it. Graphics cards (GPUs) are now the most widely used hardware for training neural networks, including NMT models, thanks to the support for heavily parallelized matrix multiplication operations.

Today, GPUs are the default option for training, along with tensor processing units (TPUs). However, the decoding can take place on many different devices, depending on the use of the actual translation model. For example, backtranslation and knowledge distillation where we translate large amounts of data usually runs on a GPU, whereas the translation of a single sentence with your mobile app runs either on a cloud (where it can use both CPUs or GPUs), or locally on your device's CPU.

There is also the option of training (and/or decoding) the models on multiple GPUs, which greatly speeds up the process. Perhaps the most common scenario is *synchronous training*. At the beginning, the model is copied across all the available devices, the gradients are computed for a batch of data on each device, then averaged (weighted by the batch size on each device, if different), and the update is performed on all copies. On the other hand, in *asynchronous training*, the process does not wait for the computation to finish on all devices, but rather performs the updates of the shared parameters as the gradients are computed. Synchronous training has the advantage of being accurate and deterministic, whereas asynchronous training converges slower (more steps are needed) but the computational resources are used more efficiently (no need to wait for slower devices). The delayed update, discussed in the batching section above, serves as a simulation of training on multiple devices.

## 2.4 Autoregressive Decoding

The decoding from a trained model is formulated as a search problem to find a target language sentence $y^*$ given the output probability distribution $p$ estimated by the translation model with parameters $\theta$ and the input sentence $x$:

$$y^* = \operatorname*{argmax}_y p(y|x, \theta) \tag{2.40}$$

The size of the search space grows exponentially with the target sentence length, making exhaustive search for the best-scoring sentence intractable.[2]

[2]Even despite the fact that during decoding the maximum output length is limited to prevent endless loops.

A simple approximation of Equation 2.40 is *greedy* decoding. Instead of searching for the global maximum, greedy decoding takes the most likely word $y_t^*$ in each step $t$:

$$y_t^* = \underset{y_t \in \mathcal{V}}{\operatorname{argmax}}\, p(y_t | y_{<t}, x, \theta) \tag{2.41}$$

Greedy decoding is a fast algorithm and is useful, for example, for validation, when we care more about the difference in performance between checkpoints than about the best results.

**Beam Search.** A widely adopted technique for exhaustive search approximation is the *beam search* algorithm. The algorithm keeps track of a small number $b$ of hypotheses. In each time step, each hypothesis is expanded with $b$ most likely tokens as predicted by the model, forming $b^2$ candidate hypotheses. Out of the $b^2$ candidates, the algorithm selects $b$ best scoring hypotheses that will form the beam in the next step. To compensate for the fact that shorter sentences receive higher probabilities, length normalization is usually taken into account when computing the hypothesis score (Wu et al., 2016):

$$\begin{aligned} s(y|x) &= \log p(y|x, \theta) / lp(y) \\ lp(y) &= \frac{(5 + T_y)^\alpha}{(5+1)^\alpha} \end{aligned} \tag{2.42}$$

where $s(y|x)$ is the score of the hypothesis $y$ given the source sentence $x$, $T_y$ is the length of $y$ and $\alpha$ is a hyperparameter that controls the strength of the length normalization.

**Ensembles and Weight Averaging.** A common technique to increase model performance during decoding is model *ensembling*. In ensembling, multiple models with the same architecture but different random initialization are trained and during decoding their output distributions are combined:

$$p_{\text{ens}}(y_t | y_{<t}, x, \theta_1 \dots \theta_n) = \frac{1}{n} \sum_{i=1}^{n} p(y_t | y_{<t}, x, \theta_i) \tag{2.43}$$

An alternative to ensembling a group of independently trained models is to use the same model from different points in time during training, called *checkpoints*. Unlike ensembles, weight averaging combines the parameters of the selected checkpoints into a single model, so the decoding is faster. Usually, checkpoints are col-

lected periodically during validation. There are a few different strategies for selecting the checkpoints, for example, taking $n$ most recent checkpoints, or $n$ best scoring checkpoints according to an automatic evaluation metric such as BLEU or cross entropy.

**Training vs. Inference.** The models we described in Sections 2.2.1 and 2.2.2 are *autoregressive* – the output tokens are predicted left-to-right, while every decision is conditioned on the previously generated outputs. With this property comes an important distinction in behavior between training and decoding. Whereas during training, the ground-truth data are used to simulate the previous decisions, during decoding, the ground truth is unknown and therefore the model needs to rely solely on its own decisions. As we discuss in Section 2.3, this constitutes a theoretical problem, called *exposure bias*, where the model is never exposed to its own errors during training.

In RNN-based models, the difference between training and decoding is minimal. Model execution is done the same way, with the exception of providing ground-truth data during training. Another exception is that the final softmax does not need to be computed during greedy decoding (there is no need for normalized distribution if we are interested only in the token with maximum probability), but is still needed for beam search.

The Transformer models are quite different in this aspect. Since there is no recurrence operation which requires accumulation of information in a hidden state, the network can be trained on a whole ground-truth sentence in one step. The only requirement is to prevent the decoder self-attention from attending to the future positions, which is achieved by the causal attention mask. However, the decoding process remains autoregressive.

## 2.5 Evaluation

The problem of MT evaluation is almost as challenging as MT itself. The most reliable method to assess the quality of MT systems remains human evaluation. Since the adoption of statistical approach for MT (Brown et al., 1993; Koehn et al., 2003), the demand for automatic translation quality metrics grew larger, as the models often need to be validated several times during training.

Perhaps the best-known automatic MT metric is BLEU (Papineni et al., 2002). Despite a long-term effort led by the organizers of the Metrics Shared Task on the Conference on Machine Translation (WMT) to create an automatic evaluation metric that would be better correlated with scores assigned by humans, BLEU continues to

be the most widely used metric in the contemporary literature. However, over the nearly two decades of using BLEU, it has been criticized for being prone to errors due to outliers or being too inaccurate when the score itself is low (Callison-Burch et al., 2006; Bojar et al., 2010; Reiter, 2018; Mathur et al., 2020; Kocmi et al., 2021).

Character F-score (ChrF) is another evaluation metric proposed by (Popović, 2015). Unlike BLEU, it is based on character-level n-gram matching. More recently, the COMET framework (Rei et al., 2020) has been proposed as a class of trainable NN-based evaluation metrics for MT. The evaluation models are trained on human evaluation judgements, and thus the evaluation metrics should mimic human behaviour.

In addition to the translation quality, the models can also be evaluated in terms of efficiency. Decoding speed is usually measured in two dimensions: *latency* and *throughput*. Latency is the time to decode a single sentence in online mode (i.e., with batch size of 1). Throughput is the maximum number of sentences that the model can translate in a unit of time (with any batch size). In the WMT Efficiency Shared Task, model size and RAM usage is also reported.

# 3

# Non-Autoregressive NMT

In real-world applications of machine translation (MT), efficiency is often crucial. Most commercial neural machine translation (NMT) models are available through a cloud-based service, such as Microsoft Translator[1] or Google Translate.[2] Scaling cloud-based solutions for large user bases is simple but costly. Even with a large pool of computational resources, it is worthwhile to implement optimizations that decrease model latency and improve user experience.

Locally deployed NMT models provide a number of advantages over cloud-based solutions. First, the service does not rely on the internet connection. Second, the data is not being sent to a third party server and, therefore, it is suitable for translating private or confidential data. However, without optimization, running state-of-the-art translation models locally often requires specialized hardware, such as one or more GPUs. Otherwise, the time to translate a single sentence can easily exceed one second on a standard CPU.

Higher decoding speeds can be achieved by model optimization. In their 2019 submission to the Workshop on Neural Generation and Translation (WNGT) Efficiency Shared Task, Kim et al. (2019) successfully employed knowledge distillation, quantization, shortlisting (Jean et al., 2015) and a simpler recurrent unit design to bring the throughput of a translation model up to 3,600 words per second on a CPU, with a modest drop in the translation quality. Following this work, Bogoychev et al. (2020) reported further improvements with attention head pruning (Voita et al., 2019). Their work has been part of the Bergamot Research Project, which aims to bring offline translation models to a browser.[3]

[1] `https://microsoft.com/translator/`
[2] `https://translate.google.com/`
[3] `https://browser.mt/`

Non-autoregressive (NAR) models present an alternative approach to model optimization, using different architecture and a different decoding algorithm which has lower time complexity. In NMT, a non-autoregressive decoding algorithm does not access previously decoded outputs, imposing conditional independence assumption on the output token probability distributions. This assumption allows for parallelization of the decoding, which can significantly reduce the latency of the translation system. On the other hand, it also presents a challenge to the language model, which usually leads to poorer translation quality.

The research presented in this thesis is focused on the usefulness of NAR models for translation (NAT) in the quest for faster translation models. We first analyze the NAT models alone and assess the necessary techniques needed to match the quality of autoregressive (AR) models. Then, we apply lexical shortlists, an optimization technique successfully used in AR models, to NAR models and evaluate the additional speed gains (see Chapter 5).

In this chapter, we present an overview of the current state of the research and key concepts in non-autoregressive NMT. We begin with the description of the general, commonly used principles of NAT in Section 3.1. Then, we provide a survey of notable approaches to NAT, focusing on extensions to the basic models that improve output quality. We categorize them into four groups, each of which is described in a separate section: First, the methods based on relaxing the alignment between the output and target sequences, which are most similar to our work in Chapters 4 and 5, are described in Section 3.2. Methods to improve the training process using auxiliary objectives are described in Section 3.3. Section 3.4 introduces approaches based on iterative decoding. Relevant methods that do not fall clearly into any of the categories above are summarized in Section 3.5. Naturally, the groups are not mutually exclusive, and there is an overlap. We conclude this chapter with a discussion of the limitations of the presented literature, mainly related to the evaluation methodology.

## 3.1 Non-Autoregressive Models

This section summarizes the main features of NAR models, some of the basic concepts, and research problems in this field. The NAR variant of the Transformer model was first described by Gu et al. (2018) and Lee et al. (2018). Each of these two foundational papers provides different (complementary) grounds for further research discussed later in this chapter – latent variables and iterative decoding.

**Conditional Independence.** The defining feature of a non-autoregressive model is the assumption of conditional independence between the output distributions across time steps. Recall Equation 2.32, which defines the output distribution for autoregressive models:

$$p(y|x) = \prod_{t=1}^{T_y} p(y_t|y_{<t}, x, \theta) \tag{2.32}$$

Unlike Equation 2.32, NAT models do not condition the output token probabilities on previously decoded outputs $y_{<t}$. The probability of an output sentence $y$ given an input sequence $x$ can then be modeled as:

$$p(y|x) = \prod_{t=1}^{T_y} p(y_t|x, \theta) \tag{3.1}$$

Although technically possible, making the outputs in RNN-based models conditionally independent does not reduce the time complexity because in RNNs, the value of each hidden state depends on the value of the preceding state (see Section 2.2.1). However, in the Transformer model (see Section 2.2.2), hidden states in each layer depend only on the states from the previous layer. This allows for parallel computation at the layer level.

In the following paragraphs, we discuss the necessary alterations to the Transformer architecture. Since the outputs are conditionally independent, we cannot feed the previously decoded outputs into the Transformer decoder. We need to provide the input to the decoder and estimate the target length. The causal mask over decoder self-attention is now unnecessary. We also address the main issue and the reason AR models are still superior in modeling language.

**Multimodality Problem.** In one of the first applications of a non-autoregressive model to NMT, Gu et al. (2018) describe the *multimodality problem* which arises when the outputs are conditionally independent.

When estimating the probability of a word on a given position, there may be multiple words which get a high probability. These words are the so-called *modes* of the distribution. In autoregressive models, once a word is selected, other modes are ignored in the following time steps. However, a non-autoregressive model does not base its decision for a given position on the preceding ones, so when multiple positions have multiple modes, the model has no means of coordinating the selection of modes across different time steps.

A well-known example of the multimodality problem is the translation of the sentence "thank you" into German, which has two equally likely translations: "vielen dank" and "danke schön." In this case, the pair of German tokens "danke" and "vielen" create the two modes in the first position, and the tokens "dank" and "schön" are the modes in the second position. If an autoregressive model chooses to generate "danke" in the first position, the token "dank" in the second position will no longer receive high probability from the model. However, when a non-autoregressive model assigns high probabilities to the correct translations, it also has to assign high probabilities to the other (incorrect) two combinations, "danke dank" and "vielen schön" (Gu et al., 2018).

**Target Length Estimation.** In a standard AR NMT model such as the Transformer or an RNN-based model with attention, the length of the output sentence is modeled implicitly by using the special end-of-sentence (`</s>`) token. Equations 2.32 and 3.1 work with the sentence length implicitly, assuming that the last token $y_{T_y}$ is the `</s>` token and does not appear among the previous tokens $y_{<T_y}$.

The probability distribution in Equation 3.1 can be factorized to express the estimation of the target sentence length explicitly:

$$p(y|x,\theta) = p_L(T_y|x,\theta) \cdot \prod_{t=1}^{T_y} p(y_t|x,\theta). \tag{3.2}$$

Explicit target length estimation is a useful technique for moderating the multimodality problem – the NAR model is conditioned on the predicted length. It also helps with the problem of supplying inputs to the decoder.

**Decoder Inputs.** A NAR Transformer decoder cannot receive the previously decoded tokens on the input. A solution proposed by Gu et al. (2018) is to use a simple fertility model, which also serves as the explicit target length estimator.

Compared to the autoregressive Transformer, the model has the following modifications. First, the inputs to the decoder are made up of the sequence of encoder inputs, either uniformly stretched to the predicted target sentence length, or copied using a fertility model. Second, the decoder self-attention does not use the causal mask, since all states can now attend to all other states in both directions. Third, a *positional attention* sub-layer is added to every decoder layer, where the positional encoding (see Equation 2.25 in Section 2.2.2) is used as queries and keys, and the decoder states as values. Gu et al. (2018) argue that providing positional information directly to the decoder layers could improve the potential of the decoder to model local reodering.

In Gu et al. (2018), the multimodality problem (and length estimation) is addressed by introducing latent fertility variables $F = f_1, \ldots, f_{T_x}$ sampled from a prior distribution. Each $f_i \in \mathbb{N}_0$ denotes the number of times $x_i$ is copied to the decoder input (summing up to the target length $T_y$). The output probability is then conditioned on the latent vector $F$, which is marginalized out:

$$p(y|x, \theta) = \sum_{F \in \mathcal{F}} p(F|x, \theta) \cdot p(y|x, F, \theta) \tag{3.3}$$

where the fertility model $p(F|x, \theta)$ and the translation model $p(y|x, F, \theta)$ can be trained jointly using a variational lower bound with a candidate distribution $q$:

$$\begin{aligned} \mathcal{L}(\theta) &= \log p(y|x, \theta) = \log \sum_{F \in \mathcal{F}} p(F|x, \theta) \cdot p(y|x, F, \theta) \\ &\geq \mathbb{E}_{F \sim q} \left( \sum_{t=1}^{T_y} \log p(y_t|x, F, \theta) + \sum_{t=1}^{T_x} \log p(f_t|x, \theta) \right) + \mathcal{H}(q) \end{aligned} \tag{3.4}$$

where $q$ is an external deterministic fertility model (and, therefore, $\mathcal{H}$ is a constant), and the expectation is also deterministic. The fertility model depends on an external module which is not trained together with the model. The authors fine-tune the trained translation model using reinforcement learning (Williams, 1992) to estimate the gradients of the fertility model.

During decoding, marginalizing over all possible fertility values is intractable. Therefore, Gu et al. (2018) experiment with three approximation methods – argmax, average decoding, and noisy parallel decoding (NPD). In argmax decoding, the fertility with the highest probability is chosen in each step, similarly to greedy decoding. The average method chooses the expected fertility given the distribution in each position. NPD is based on sampling and rescoring with an autoregressive model, as explained below.

**Autoregressive Rescoring.** A usual technique in the NAT literature is to use some form of rescoring of the outputs of the non-autoregressive model by an autoregressive model. The nature of the Transformer model allows scoring a sentence in a single step (as opposed to generating one), which means that the additional rescoring computation increases the decoding complexity only linearly with respect to the number of the candidates for rescoring.

The choice of the rescoring candidates varies. Gu et al. (2018) use NPD which first samples the decoder input sequences from the fertility distribution, computes the best candidate for each input, and then rescores the results using an AR model. A closely related method is length parallel decoding (LPD), which generates and scores sentences of different lengths.

Note that in contrast to AR models, the highest scoring sequence in an NAR model can be found simply by taking the highest scoring token in each step, because of the conditional independence assumption.

**Knowledge Distillation.** To tackle the multimodality problem from another angle, Gu et al. (2018) propose to use sequence-level knowledge distillation to create artificial training data (Kim and Rush, 2016). As explained in Section 2.3.1, knowledge distillation is based on training an AR teacher model and using it to create artificial training data for a student model. The main idea is that the outputs of a teacher model will have a simpler structure than natural language, limiting the number of distribution modes.

According to an ablation study published by Gu and Kong (2021), using knowledge distillation is a crucial element in training non-autoregressive translation models, regardless the actual method used. Zhou et al. (2020) further study the effects of knowledge distillation strategies on the translation quality of NAR models. They link the data complexity to the potential of NAR modeling and find that knowledge distillation reduces the complexity, which in turn leads to improved model performance.

**Iterative Refinement.** Modeling fertility introduces latent variables into a non-autoregressive model. A different approach was proposed by Lee et al. (2018). Instead of modeling fertility, the authors introduce $L$ discrete sequential latent variables interpreted as stages of refinement:

$$\begin{aligned} p(y|x) &= \sum_{y^L} \left( \prod_{t=1}^{T_y} p(y_t|y^L, x) \right) p(y^L|x) \\ p(y^L|x) &= \sum_{y^{L-1}} \left( \prod_{t=1}^{T_y} p(y_t^L|y^{L-1}, x) \right) p(y^{L-1}|x) \\ &\vdots \\ p(y^0|x) &= \prod_{t=1}^{T_y} p(y_t^0|x) \end{aligned} \tag{3.5}$$

Each summation in the equations above is computed over the whole space of possible sequences, and thus it is intractable to calculate the probability $p(y|x)$ exactly. To overcome this problem, the authors approximate the sums with only the element corresponding to the sequence with the highest probability, $\hat{y} = \text{argmax}_y\, p(y|x)$. Putting it together with the equations above and moving to the logarithmic domain, we get:

$$\begin{aligned}\log p(y|x) \geq & \sum_{t=1}^{T_y} \log p(y_t|\hat{y}^L, x) + \\ & + \sum_{t=1}^{T_y} \log p(\hat{y}_t^L|\hat{y}^{L-1}, x) + \dots \\ & + \sum_{t=1}^{T_y} \log p(\hat{y}_t^0|x).\end{aligned} \tag{3.6}$$

The tokens $\hat{y}_t^0$ in the initial sequence are set to $x_{t'}$ where $t' = (T_x/T_y) \cdot t$, i.e. the source sentence is either squished or stretched by copying or omitting some of the source words in order to fit the target sentence length. During training, the length of the reference sentence is known. During decoding, the authors use a separate model $p(T_y|x)$ for target length prediction. All probability distributions in the above equations are modeled with neural networks with shared parameters. In this way, the number of intermediate refinement steps during decoding can be arbitrarily chosen and remains flexible after the model is trained.

The latent variable model is trained by minimizing the log-likelihood of the reference sentence $y^*$ in each of the refinement steps:

$$\mathcal{L}_{\text{LVM}}(\theta) = -\sum_{l=1}^{L+1} \left( \sum_{t=1}^{T_{y^*}} \log p(y_t^*|\hat{y}^{l-1}, x, \theta) \right) \tag{3.7}$$

Lee et al. (2018) also discuss the training of the refinement process from a non-iterative denoising perspective. Formally, they introduce an additional denoising autoencoder loss to the model:

$$\mathcal{L}_{\text{DAE}}(\theta) = -\sum_{t=1}^{T_y} \log p(y_t^*|\bar{y}, x, \theta) \tag{3.8}$$

where $\bar{y}$ is a corrupted version of the reference translation $y^*$. The corruption process is performed on each token with a probability $\beta$. Each corrupted token is either replaced with a random word from the vocabulary, copied to its neighbor, or swapped with the neighbor.

During training, the two loss functions are stochastically mixed using a hyperparameter $\alpha$, sampled from a Bernoulli distribution. In other words, in each refinement step, the input to the decoder is either the output of the previous refinement step or a corrupted version of the reference sentence.

## 3.2 Alignment-Based Methods

Autoregressive models are trained using the cross-entropy loss (recall Equation 2.34 in Section 2.3). If we assume a single sentence pair $(x, y)$, the negative log-likelihood is the sum of negative log-probabilities $\log p(y_i|x, y_{<i}, \theta)$ for $i = 1, \ldots, T_y$, as estimated by the model with parameters $\theta$. Notice how the ground-truth sequence $y_1, \ldots, y_{T_y}$ corresponds to the sequence of probabilities $p(y_i| \ldots)$: we require that the $i$-th reference token is decoded at the $i$-th position of the output state sequence. The methods described in this section relax this requirement.

We define alignment as a function $a$ which maps the positions in the ground-truth sentence to the positions in the output. In autoregressive models, $a(i) = i$ for every position $i$ in the ground truth. Methods that consider different alignments allow us to work with output and label sequences of different lengths, which is a useful feature in the context of NAT models.

To the best of our knowledge, we were the first to propose an alignment-based method for NAT (Libovický and Helcl, 2018). Our method is based on connectionist temporal classification (CTC), which computes the cross-entropy losses over all possible alignments between the ground truth and a longer output state sequence. We describe this method in detail in Chapter 4. In the following paragraphs, we present a number of related methods based on modeling the alignment between the sequences of outputs and ground-truth labels.

**Aligned Cross Entropy for NAT.** The main problem with using cross-entropy objective for training NAT models is that it heavily penalizes misaligned target words. If a correct word is generated at an incorrect position in the target sentence, it does not have a positive effect on the loss, the same as if a completely unrelated word is generated. In autoregressive models, this problem is avoided with teacher forcing; the model is provided with the preceding words from the reference sentence. Without teacher forcing, the alignment between the predicted distributions and the positions in the reference sentence is too rigid. For example, consider the reference sentence "Thank you for your cooperation." In a non-autoregressive model, another

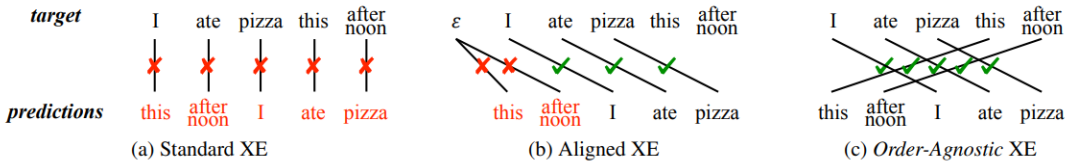
valid translation "Thanks for your cooperation" receives the same training signal as a completely wrong translation. The crucial difference in AR models is that to train the prediction at the second position, the target word "Thank" for the first position is provided on the input, so the word "for" does not receive high probability.

One way to address this problem is to consider the alignment as a latent variable. The aligned cross-entropy (AXE; Ghazvininejad et al., 2020a) objective function uses a dynamic programming algorithm to find the alignment with the minimum cross-entropy loss. In the example from the previous paragraph, the alignment with the lowest cross entropy is the one that aligns the positions 3–5 from the label sequence (representing the suffix "for your cooperation") to the output positions 2–4 in the output. Either of the first two positions in the label sequence can be aligned to the first position in the output sequence, while the other label position remains unaligned.

The alignments considered in the AXE approach are monotonic (i.e., $a(i) \leq a(j)$ for every $i \leq j$, when defined), but the mapping does not have to be defined for each ground-truth position. Similarly, not all positions in the prediction sequence are not required to be mapped to a reference position. In case of skipping a prediction, the model learns to output an empty token $\epsilon$, i.e. $-\log p(y_t = \epsilon|x, \theta)$ is added to the loss. When a position in the ground-truth sequence is skipped, the negative log-probability of the skipped token is added to the loss at the current prediction position, weighted by a hyperparameter $\delta$ which controls the cost of the skip-target operation. An example of such alignment is shown in Figure 3.1 (b). The alignment with the minimum cross-entropy is chosen for computing the gradients and updating the model.

The authors use conditional masked language models as the base architecutre (described later in Section 3.4). Since any empty tokens produced are discarded during decoding, the authors choose a similar approach to Libovický and Helcl (2018) and initialize the decoder input length with the source length multiplied by a factor of $\lambda$, which they estimate empirically from the data. We present this approach in detail in Chapter 4.

In the experiments described in Chapter 4, we use the CTC loss, which is similar to this approach. Instead of considering the alignment that yields the minimum cross-entropy loss, the CTC algorithm computes the sum of all possible alignments. Unlike AXE, we only consider skipping predictions; we do not allow the model to skip the target positions.




(a) Standard XE (b) Aligned XE (c) *Order-Agnostic* XE

Figure 3.1: An illustration of (a) cross-entropy, (b) aligned cross-entropy, and (c) order-agnostic cross-entropy on a toy example. We use the figure from Du et al. (2021), Figure 1

**Order-Agnostic Cross-Entropy.** The alignment-based method discussed in the previous paragraphs considers only monotonic alignments. Du et al. (2021) design order-agnostic cross-entropy (OaXE), a new training objective which also permits non-monotonic alignments (see illustration in Figure 3.1). Similarly to AXE, they select the ordering with minimum cross-entropy for gradient propagation.

The dynamic programming algorithm used in the CTC and AXE methods is no longer suitable for non-monotonic alignments. Moreover, the search space is even larger than in the previous case, as it is proportional to the factorial of the target length. The authors use the Hungarian algorithm (Kuhn, 1955), which finds a maximum bipartite matching representing the alignment in polynomial time. They also incorporate heuristics and tweak the search algorithm to exclude invalid orderings or predictions.

**Imputer.** Another variant of the alignment-based approach was described by Saharia et al. (2020). They experiment with a CTC-based model similar to the model we use in Chapter 4, and an iterative variant called *Imputer*.

The model architecture is a stack of self-attentive layers without the causal mask. The CTC-based model takes upsampled source sentence embeddings (in order to be able to create translations that are longer than the source) and processes them with the Transformer layer stack. The Imputer model proceeds in a fixed number $L$ of iterations. In each iteration, the model generates ("imputes") $\left\lceil \frac{T_y}{L} \right\rceil$ at some positions where $T_y$ is the output length. This procedure is similar to decoding from masked language models described in Section 3.4. In addition to the CTC model, the input to the Imputer model is the embedding of the partially decoded sentence from the previous iteration summed with the (upsampled) source sentence embeddings.

The training of the Imputer model takes into account every partial output potentially already decoded and maximizes a log-likelihood lower bound, which can be comptued with a dynamic programming algorithm.

## 3.3 Auxiliary Training Objective Methods

Auxiliary training objectives can be viewed as a form of regularization of the training process. Unlike techniques that aim at improving convergence speed, such as $L_2$ regularization or dropout, the objectives described here are designed to help non-autoregressive models cope with the conditional independence assumption in order to better model the target language.

The following methods are based on including one or more auxiliary objectives in the training procedure. Unlike the methods in Sections 3.2, 3.4 and 3.5 which also use auxiliary objectives, none of these methods changes the decoding process: Decoding simply takes the highest scoring token for each time step in parallel.

**NAT with Auxiliary Regularization.** Wang et al. (2019) analyze the basic NAT model (Gu et al., 2018) and identify two frequent translation errors – repeated and incomplete translation. For each of these symptoms, they design a special auxiliary training objective to mitigate their effect.

For the repeated translation problem, where the same token is generated multiple times in conseutive time steps, the authors propose to increase the dissimilarity of adjacent hidden states when different tokens should be decoded, tying the similarity of the target token embeddings to the similarity of the corresponding hidden states.

The incomplete translation problem is a case of target tokens missing from the predicted translation. To address this problem, the reverse AR translation model is used to reconstruct the source sentence back from the NAR decoder states. The loss of this backward translation model is again used as an auxiliary objective.

In the final NAT-REG model of Wang et al. (2019), the two auxiliary losses are mixed in with the standard cross-entropy loss, using weight hyperparameters which are set empirically.

**Hint-NAT.** As mentioned in the introductory section of this chapter, knowledge distillation is often used in NAT research. Li et al. (2019) decide to use an AR teacher model to provide signals (hints) to the training of the NAR student model.

They propose using two types of hints from the teacher model. First, to bring the values of the teacher and student decoder hidden states closer together, the authors consider tying the states with $L_1$ or $L_2$ distance, but argue that this straightforward approach destabilizes the student training process and fails. Instead, they use a weaker signal from the teacher model and define an auxiliary loss based on the cosine distances within the decoder hidden states. Second, they use the information from the teacher model cross-attention to guide the alignment learning in the student

model. Again, this information is incorporated into the training as an auxiliary loss, based on the KL divergence between the teacher and student attention distributions. Again, the auxiliary losses are mixed using weighting hyperparameters together with the cross-entropy loss.

**Bag-of-ngrams Difference.** Shao et al. (2020) address the issue of misalignment between the targets and the outputs. They define the bag-of-ngrams (BoN) difference between the model output distributions and the target sentence, and instead of finding an appropriate alignment, they design a new objective to minimize the BoN difference.

The BoN objective is based on the $L_1$ distance between two sparse vectors representing the n-grams that are present in the reference sentence and are likely to be present in the translation (based on the output probabilities). As in the previous two cases, this objective is complemented by the cross-entropy objective. The authors also show that fine-tuning with the BoN objective alone is helpful.

**Glancing Training.** Qian et al. (2021a,b) propose Glancing Transformer (GLAT), a technique for training the NAT model similarly to conditional masked language models (CMLMs), but enabling decoding in a single pass instead of an iterative process.

During training, the model first generates an intermediate output sentence non-autoregressively, without computing the loss. Then, a number of positions is masked according to a glancing sampling strategy, and the model is trained to predict the masked tokens on the selected positions. The sampling strategy randomly selects $S$ positions, where $S$ is proportional to the number of errors in the intermediate prediction.

The decoding is similar to the parallel process proposed by Gu et al. (2018) described in Section 3.1, but instead of a fertility model, the target sequence length prediction is done using the output representation of an artificial `LENGTH` token which is added to the decoder input (as in the CMLM approach of Ghazvininejad et al., 2019, see Section 3.4).

## 3.4 Iterative and Semi-Autoregressive Methods

The common theme in all of the approaches described so far is that during inference, the underlying Transformer model is executed only once. Autoregressive models, on the other hand, need $T_y$ consecutive executions (passes through the network) to generate a sentence of $T_y$ tokens. This linear relationship follows from the conditional dependency (see Equation 2.32 in Section 2.2.2).

In this section, we look at methods that lie somewhere in between. In these approaches, the number of model executions is greater than one, but it is still constant with respect to the target length $T_y$. We call these approaches *iterative*. We also include approaches which operate in a fraction of $T_y$ steps, even though the number of steps is no longer constant. We refer to these approaches as *semi-autoregressive*.

Many of the methods described here are based on the ideas of iterative refinement first introduced by Lee et al. (2018), which we have already reviewed in Section 3.1.

**Iterative Decoding from Masked Language Models.** Recently, pre-trained masked language models (MLMs) such as BERT (Devlin et al., 2019) have attracted attention from the translation research community. Ghazvininejad et al. (2019) introduce conditional masked language models (CMLMs), an extension to cross-lingual language models (XLMs; Conneau and Lample, 2019) for multilingual applications, including MT.

Conventional language models (LMs) estimate the probability of a word given the preceding words. In contrast, MLMs are trained on sentences where a number of tokens have been masked out, and the objective is to predict what the masked words were, given the rest of the sentence. XLMs extend this idea to a pair of sentences in two (or more) languages. That is, tokens from either the source or the target sentence, or both, are masked and the model tries to predict them. To facilitate alignment modeling and language identification, XLM architectures include a language embedding and reset the positional encodings for each sentence. See Figure 3.2 for an illustration of the monolingual and cross-lingual LM training objectives.

Compared to the original formulations of XLMs, CMLMs differ in a few aspects: First, CMLMs use an encoder-decoder architecture, rather than a decoder-only model that works on the concatenation of the source and target sentences. Second, only the target words are masked and subject to training and prediction. CMLMs are also intended to be used directly for language generation, rather than for cross-lingual LM pretraining, as is the case of XLMs.

The decoding process (called *Mask-Predict*) starts with the estimation of the target length. For this purpose, Ghazvininejad et al. (2019) include a special `LENGTH` token in the decoder input and predict the target length as the output on this position. This way the length predictor can be trained jointly with the translation model using the cross-entropy training signal. The authors show that it is useful to sample a small number of different length candidates, decode a translation for each length, and then pick the highest scoring one.

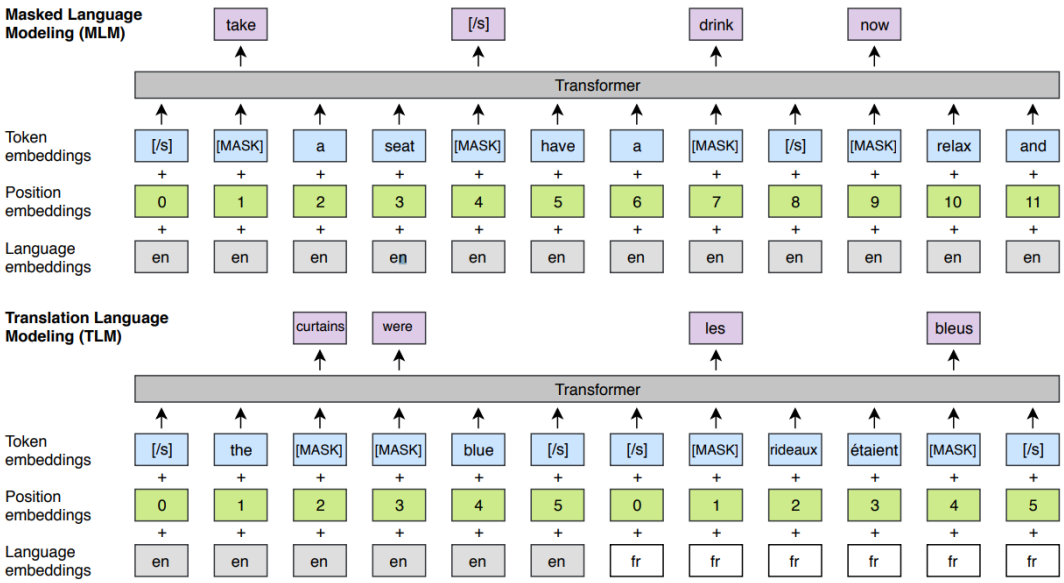



Figure 3.2: A comparison of masked language model and translation language model training objectives for training cross-lingual language models. Image source: Conneau and Lample (2019), Figure 1.

Given a target length, the generation begins with a sequence of mask symbols. In each iteration, the tokens on all masked positions are predicted in parallel (independently, and thus non-autoregressively). Then, a number of the tokens that receive the lowest probability by the model are selected and masked-out again, prepared to be re-predicted in the next iteration. The number of masked tokens decreases with each iteration, so a constant number of iterations is needed to generate the whole translation.

In their generalized framework for sequence generation, Mansimov et al. (2019) experiment with other masking strategies for CMLMs, such as left-to-right, easy-first, or uniform.

**Semi-autoregressive training.** In their follow-up experiments, Ghazvininejad et al. (2020b) address the exposure bias issue in the context of CMLMs As discussed in Section 2.3, the exposure bias problem arises when the model is trained on ground-truth data, but during decoding, it relies on its own (often incorrect) decisions instead. CMLMs are trained to predict masked tokens given the ground-truth context, but during decoding, the refinement is done using generated context instead.

To tackle this issue, Ghazvininejad et al. (2020b) propose to mix the ground-truth tokens with the model predictions. This approach is somewhat similar to scheduled sampling for recurrent neural networks (Bengio et al., 2015).

**Disentangled context.** Kasai et al. (2020) extend the CMLM model with a few additions. Instead of predicting only the masked tokens, the authors propose an architecture that is able to predict a subset of the *other* tokens for each position in the target sequence. They achieve this goal by adapting the attention module. First, the self-attention does not attend to its own position in the sequence on the previous layer. Second, to prevent information leakage between the same positions on different layers, the self-attention keys and values are decontextualized, meaning that instead of hidden states from the previous layer, the decoder input embeddings are used.

**Jointly masked model for NAT.** Following the practice from training the BERT model (Devlin et al., 2019), Guo et al. (2020) mask source tokens in the encoder in each step to achieve a more robust encoder representation. They introduce an auxiliary loss in the encoder to predict the masked tokens.

Instead of masking out random tokens in the decoder, the authors propose to mask whole random n-grams. They also use an additional auxiliary n-gram loss tailored to help with the problem of repeated tokens in the translation (see Section 3.3). The Mask-Predict strategy (Ghazvininejad et al., 2019, see above) is employed during inference.

**Semi-Autoregressive NMT.** One of the first attempts to bring together the best of the worlds of AR and NAR models was proposed by Wang et al. (2018). They use a semi-autoregressive Transformer model, which predicts groups of $K$ consecutive tokens at a time. The tokens in each group are predicted independently and in parallel, but the conditional dependency is retained between the groups in an autoregressive fashion.

As expected, the authors find that increasing $K$ leads to degraded translation quality but brings improvements in terms of decoding speed.

**Blockwise Parallel Decoding for Deep Autoregressive Models.** Stern et al. (2018) propose a similar semi-autoregressive approach where chunks of the target sentence are generated in parallel. They start with greedy decoding from an autoregressive model, $p_1$, and introduce additional "look-ahead" models $p_2, \dots p_k$. In time step $t$, each model $p_i$ predicts the $(t + i)$-th word in the target sequence given the same prefix of $t$ previously decoded words.

The decoding process has three stages. First, the block of predictions is computed using the models $p_1, \ldots, p_k$. Second, model $p_1$ is used to verify the $(k-1)$ candidates (which is done in parallel in the Transformer model) and finds the largest $\hat{k}$ such that teh decoded words from models $p_i$, $1 \leq i \leq \hat{k}$ are all considered best by $p_1$. Third, the accepted $\hat{k}$ words are generated, and the decoding process moves to time step $t + \hat{k}$.

**Insertion Transformer.** The Insertion Transformer (Stern et al., 2019) is an extension of the Transformer model which handles sequence generation in an arbitrary order. Instead of predicting tokens left-to-right, the model predicts sequences of insertion operations. An insertion operation specifies which tokens to insert and where to insert them. In each step, the insertion operations are generated in parallel and independently; we can thus include this approach in the semi-autoregressive category. The authors experiment with decoding in a balanced binary tree order, which can generate the output sequence in a logarithmic number of steps.

**Levenshtein Transformer.** An interesting combination of the ideas introduced in the Insertion Transformer and CMLMs is the Levenshtein Transformer (LevT; Gu et al., 2019). This model operates in an iterative fashion, much like the aforementioned methods. At the beginning, LevT takes either an empty sentence or a crude, intermediate sentence for refinement. Each iteration consists of applying three policies. First, the tokens to be deleted are identified and removed from the sequence. Second, the sentence is expanded by placeholder symbols. For each slot (a position between two tokens), the model decides how many placeholders to insert. Third, for each placeholder, a token from the vocabulary is selected to replace it, resulting in a refined version of the input sentence. Each of the three stages of a refinement iteration is performed non-autoregressively.

**Latent Transformer.** Kaiser et al. (2018) use discrete latent variables in a model called Latent Transformer (LT). The model has three components: First, the *autoencoder* takes a sentence pair $(x, y)$ and encodes the target sentence $y$ into a sequence of discrete latent variables $l$. Second, the autoregressive *latent prediction* model predicts the sequence $l$ based on the source sentence $x$. Third, a non-autoregressive *decoder* generates $y$ given the source sentence $x$ and the hidden sequence $l$.

The autoencoder is used to condense the longer target sequence $y$ into a shorter sequence (usually 8 times) $l$, which serves as an intermediate representation of $y$. Unlike perhaps more common autoencoders, this autoencoder uses discrete latent variables. A stack of convolutions with residual connections is selected as the autoencoder architecture. The latent prediction model is a stack of Transformer layers, and the decoder consists of deconvolutions.

## 3.5 Other Methods

In this section, we summarize the related work that does not fall clearly within any of the three categories from the above sections.

**Dynamic-transition conditional random fields.** In the original NAT model, the generation of the output sentence is done by selecting the tokens with the highest probability at each position. Instead of the argmax decoding, Sun et al. (2019) propose to use conditional random fields (CRF), which allows the modeling of causal relations within the output sequence. Although this process is not non-autoregressive given the nature of the CRF decoding, the Transformer model needs to be run only once and does not need to take into account the tokens as they are being generated.

In Section 4.4 we present a similar idea based on a CTC-based model, using rescoring with an n-gram language model.

**Flowseq.** Ma et al. (2019) propose to incorporate an intermediate latent variable $z$ into the model. The authors argue that the prior distribution $p(z|x)$ needs to be complex to model all dependency relations among the target tokens. They propose to use generative flow (Rezende and Mohamed, 2015) for deriving the complex distribution from a simple prior using a series of invertible transformations.

**Iterative knowledge distillation.** The common way of addressing the multi-modality problem is knowledge distillation using an autoregressive teacher (see Section 2.3.1). Sun and Yang (2020) propose to iterate the knowledge distillation step. They alternate the training of the AR teacher model and the NAR student model in an EM fashion, where the selection of samples for the AR training dataset is informed by the log-likelihood as modeled by the NAR model. In this way, the level of multi-modality in the distilled corpus is lower, and thus NAR models trained on this corpus should perform better.

The decoding from the NAR student model is performed in a single step. Additionally, the authors propose optimal deduplicated decoding (ODD), which tackles the issue of repeated outputs while preserving the predicted output length and finds the optimal sequence without repetition.

**ReorderNAT.** Ran et al. (2021) introduce an intermediate Transformer block to model a reordered source sentence for translation to the target language.

Formally, the translation model $p(y|x)$ is altered by including a latent variable $z$ (a sequence of source token embeddings on new positions) which is marginalized out:

$$p(y|x) = \sum_z p(z|x)p(y|z,x) \tag{3.9}$$

where $p(z|x)$ is the reordering module, and $p(y|z,x)$ is a NAR Transformer decoder that receives the sequence $z$ on the input. The authors experiment with both AR and NAR variants of the reordering module, concluding that using a single-layer autoregressive Transformer decoder performs best while still retaining speed improvements.

The reordering module is trained in a supervised fashion using the `fast_align` alignment tool (Dyer et al., 2013). The model is trained end to end by minimizing the sum of the reordering and decoder module losses.

**Layer-Wise Prediction with Deep Supervision.** A recent approach by Huang et al. (2021) uses grounding of the Transformer hidden states in all layers to the target sentence. On every decoder layer, the model estimates the output probability distributions from the hidden states using a linear projection and a softmax layer. From these layer-level output distributions, the embeddings of the most probable tokens are added to the input to the next layer along with the hidden state values.

The deep supervision grounds the states of all layers, not just the last one, in the ground-truth target sequence. The objective function is extended to include the cross entropies of the layer-wise softmax predictions on all Transformer layers in the decoder stack.

## 3.6 Discussion

In the conclusion of this chapter, we take a high-level view of the approaches we have presented here. We point out a few aspects that most of the literature has in common, as well as some issues regarding the evaluation and comparison to meaningful baselines.

**Results.** Table 3.1 shows selected results of the described approaches in English $\rightarrow$ German MT, measured on the WMT 14 test set, which is considered the standard benchmark. From each paper, we select the best result in terms of BLEU, and a result of a purely non-autoregressive variant (i.e., without rescoring or additional iterations), if applicable.

We can see that rescoring using an autoregressive teacher consistently helps the translation performance. However, the more rescoring steps are taken, the less decoding speed is gained.

**Capturing Reordering.** Many papers referenced in this section claim that non-autoregressive models cannot capture reordering, which is seen as the main culprit of the low translation quality of the vanilla NAT model (Gu and Kong, 2021; Ran et al., 2021). While this might be true to some extent, we see the multimodality problem as the root cause. We argue that reordering itself can be modeled by stacks of self-attentive layers, and if the space of possible target sentences is less multimodal, reordering is not a big problem for a NAT model, as illustrated by Du et al. (2021).

However, the effect of reordering on data complexity is intuitive. Zhou et al. (2020) view the amount of reordering in a translation dataset as a predictor of its complexity and show that knowledge distillation reduces the reordering score, leading to a simpler dataset, and thus reducing the multimodality.

**Translation Quality.** Weak autoregressive baseline models are a common element in the non-autoregressive literature. In most cases, the base variant of the Transformer model is used as baseline (and most of the base hyperparameters are used for the NAT model as well), arguing that having similar numbers of parameters makes the baseline and the NAT model comparable. A comparable model size is a valid point; on the other hand, it raises doubts about the scalability of the proposed approaches; the difference between large AR and NAR model variants might not be proportional.

Moreover, even the Transformer base model can be trained in a better way than as reported by Vaswani et al. (2017) and as referenced by many NAT papers as their baseline, achieving a better translation quality (Popel and Bojar, 2018). A reasonable improvement of the baseline could be achieved by training a student AR model on the knowledge-distilled data.

**Decoding Speed.** Similarly to the previous paragraph, the baselines that appear in the literature are usually weak also in terms of decoding speed. For example, Gu et al. (2018) report a latency of 408 ms for their AR model (a Transformer base), measured on a single Nvidia P100 GPU, without batching. In contrast, our implementation achieves a latency of around 100 ms with the same model under the same hardware and hyperparameter settings and no optimizations. Additionally, some articles cite a baseline AR latency of 607 ms (Wang et al., 2019; Guo et al., 2020), which is the result of Gu et al. (2018) with beam search, even though beam search is not used in the presented NAR models for the most part.

| | Method | En → De | De → En |
|---|---|---|---|
| | Gu et al. (2018) | | |
| | NAT + FT | 17.69 | 21.47 |
| | NAT + FT + top-100 NPD rescoring | 19.17 | 23.20 |
| | Lee et al. (2018) | | |
| | 1 iteration | 13.91 | 16.77 |
| | 10 iterations | 21.61 | 25.48 |
| Alignment | Ghazvininejad et al. (2020a), AXE CMLM | 23.53 | 27.90 |
| Alignment | Du et al. (2021), CMLM + OaXE | 26.10 | 30.20 |
| Alignment | Saharia et al. (2020) | | |
| Alignment | CTC | 25.70 | 28.10 |
| Alignment | Imputer | 28.20 | 31.80 |
| Alignment | Gu and Kong (2021), NAT + CTC + GLAT | 27.20 | 31.39 |
| Auxiliary Objectives | Wang et al. (2019) | | |
| Auxiliary Objectives | NAT-REG, no rescoring | 20.65 | 24.77 |
| Auxiliary Objectives | NAT-REG, top-9 rescoring | 20.65 | 24.77 |
| Auxiliary Objectives | Li et al. (2019) | | |
| Auxiliary Objectives | Hint-NAT, no rescoring | 21.11 | 25.24 |
| Auxiliary Objectives | Hint-NAT, top-9 rescoring | 25.20 | 29.52 |
| Auxiliary Objectives | Shao et al. (2020), BoN-Joint+FT | 20.90 | 24.61 |
| Auxiliary Objectives | Qian et al. (2021a) | | |
| Auxiliary Objectives | GLAT + CTC | 26.39 | 29.54 |
| Auxiliary Objectives | GLAT + CTC + top-7 NPD rescoring | 26.55 | 31.02 |
| Iterative | Ghazvininejad et al. (2019) | | |
| Iterative | CMLM + Mask-Predict, 1 iteration | 18.05 | 21.83 |
| Iterative | CMLM + Mask-Predict, 10 iterations | 27.03 | 30.53 |
| Iterative | Kasai et al. (2020), DisCo + Easy-First | 27.34 | 31.31 |
| Iterative | Guo et al. (2020) | | |
| Iterative | JM-NAT, 4 iterations | 27.05 | 31.51 |
| Iterative | JM-NAT, 10 iterations | 27.69 | 32.24 |
| Iterative | Kaiser et al. (2018) | | |
| Iterative | LT, no rescoring | 19.80 | – |
| Iterative | LT, top-100 rescoring | 22.50 | – |
| Other | Sun et al. (2019) | | |
| Other | NAT + DCRF, no rescoring | 23.44 | 27.22 |
| Other | NAT + DCRF, top-19 rescoring | 26.80 | 30.04 |
| Other | Ma et al. (2019) | | |
| Other | FlowSeq-large | 23.72 | 28.39 |
| Other | FlowSeq-large + top-30 NPD rescoring | 25.31 | 30.68 |
| Other | Sun and Yang (2020), EM+ODD | 24.54 | 27.93 |
| Other | Ran et al. (2021) | | |
| Other | AR reordering | 26.49 | 31.13 |
| Other | NAR reordering | 22.79 | 27.28 |
| Other | Huang et al. (2021), CTC + DSLP | 27.02 | 31.61 |

Table 3.1: The results (in terms of BLEU) of the described models measured on the WMT 14 test set, as reported by the authors.

| Publication | GPU type | CPU? | Batch |
|---|---|---|---|
| Gu et al. (2018) | P100 | ✗ | 1 |
| Lee et al. (2018) | P100 or P40 | ✓ | 1 |
| Kaiser et al. (2018) | GTX 1080 | ✗ | 1, 64 |
| Ghazvininejad et al. (2019) | Possibly V100 | ✗ | 10 |
| Sun et al. (2019) | P100 | ✗ | 1 |
| Wang et al. (2019) | P100 | ✗ | 1 |
| Li et al. (2019) | Possibly M40 | ✗ | 1 |
| Ma et al. (2019) | TITAN X | ✗ | *various* |
| Ghazvininejad et al. (2020a) | *Not reported* | ✗ | *unknown* |
| Shao et al. (2020) | TITAN X | ✗ | 1 |
| Guo et al. (2020) | GTX 1080 Ti | ✗ | 1 |
| Kasai et al. (2020) | V100 | ✗ | 1 |
| Qian et al. (2021a) | GTX 1080 Ti | ✗ | 1? |
| Ran et al. (2021) | P40 | ✗ | 1 |
| Gu and Kong (2021) | V100 | ✓ | 1 |
| Du et al. (2021) | *Not reported* | ✗ | 1 |
| Huang et al. (2021) | V100 | ✗ | 1 |

Table 3.2: The hardware setting and decoding batch size for measuring the decoding speed as reported in a sample of papers described in this section.

**Evaluation Methodology.** Another aspect we need to address is the evaluation methodology itself. Setting aside the fact that automatic quality evaluation using BLEU is the only reported metric, perhaps complemented by a cursory manual evaluation on a small sample, the evaluation of the speed improvements is inconsistent and sometimes the interpretation of the results is outright wrong.

The main problem is that the actual decoding speed depends on a lot of factors that are not easily reproducible. The most obvious factor is perhaps the hardware on which the decoding speed is measured, followed by the implementation. The average decoding time per sentence is also heavily influenced by the batch size in batched decoding. Table 3.2 shows that these conditions vary wildly within the literature.

A popular solution that takes the varying conditions into account for evaluating decoding speed is to report the relative speed-up measured between experiments within a single study. However, comparing these relative speed-up ratios between different papers disregards the actual decoding times – it is easier to achieve 20 times speed-up over a slow baseline than over a baseline which is faster.

Due to these reasons, we believe that we cannot show a fair comparison of the methods presented here on a graph showing the pareto frontier between the decoding speed and translation quality.

However, it is challenging to find a way to compare the contributions of different research groups objectively. One such effort is made by the organizers of the Efficiency Shared Task, now organized yearly at the Conference on Machine Translation (WMT; Heafield et al., 2020, 2021).

In the Efficiency Shared Task, submissions are evaluated both in terms of translation quality and decoding speed under various settings. All submissions are evaluated on the same hardware, hosted by Amazon Web Services. To amortize the effects of data and model loading, the decoding time is measured on a dataset that contains one million sentences. The speed is measured in five different scenarios: GPU latency (decoding without batching) and throughput (decoding with the optimal batch size), latency and throughput on a single CPU, and throughput on a multi-core CPU. The main findings of the shared task are that autoregressive models can be optimized to achieve latency of 5–20 ms even with inexpensive hardware, without sacrificing much in terms of translation quality (Heafield et al., 2021).

Our final observation is that measuring latency on a single GPU without batching is a setup that favors NAR models, even though other scenarios should also be considered for real-world applications. In batched GPU decoding, the differences between AR and NAR models fade, because batch-parallelization of AR models makes up for time-parallelization of NAR models. On the other hand, in an online decoding scenario on a CPU, only limited parallelization is possible, so the advantage of NAR models is also diminished. We present detailed analysis of this behavior in Chapter 5.

# 4

# Non-Autoregressive NMT with Connectionist Temporal Classification

*This chapter is based on the paper* "End-to-End Non-Autoregressive Neural Machine Translation with Connectionist Temporal Classification"*, joint work with Jindřich Libovický, published at EMNLP 2018.*

In this chapter, we lay the foundations for the non-autoregressive neural machine translation (NAT) approaches studied in this thesis. We describe our experiments with an architecture based on the connectionist temporal classification (CTC) loss (Libovický and Helcl, 2018).

We begin with the description of the CTC algorithm for computing the cross-entropy loss over all possible alignments between the reference sentence and the sequence of output states in Section 4.1. Section 4.2 introduces the proposed NAT model architecture based on the Transformer model (Vaswani et al., 2017). The results of the preliminary experiments are discussed in Section 4.3. In Section 4.4, we present our attempt to address the most severe issue non-autoregressive (NAR) models have, which is the reduced fluency compared to the autoregressive (AR) models (Kasner et al., 2020; Kasner, 2020). Section 4.5 summarizes the limitations of the presented approaches.

$$\text{a cat sat on a mat} = \begin{cases} \text{a <blank> cat sat on a <blank> mat} \\ \text{a a cat cat sat on a mat} \\ \text{a <blank> cat cat sat on a mat} \end{cases}$$

Figure 4.1: A group of output sequences of equal length which all represent the same target in CTC.

## 4.1 Connectionist Temporal Classification

CTC (Graves et al., 2006) is a method for training neural networks on sequential data. Originally applied to the phonetic labelling task, but later successfully adapted in related areas, including automatic speech recognition (ASR) or handwriting recognition (Liwicki et al., 2007; Eyben et al., 2009; Graves and Jaitly, 2014).

The main strength of CTC becomes evident in tasks where the input and output labels are weakly or not at all aligned, for example, in situations where the observed input sequence is considerably longer than the target output sequence – hence the application to ASR, where the number of extracted features per second is higher than the number of phonemes uttered per second.

Training neural networks with CTC is independent of the actual neural network architecture. The CTC loss function can be applied on any network with sequential outputs. Thus, this method is applicable to both recurrent neural networks (RNNs) and the Transformer model.

Models trained with the CTC assume that the alignment between the input states (e.g. a group of frames in an audio signal) and the output states (e.g. a phoneme) is unknown. A variable number of frames in a row can encode a single phoneme. Similarly, in translation, multiple words in the source language may correspond to any number of (even non-consecutive) words in the target language.

The idea behind CTC is to allow some states to produce no output. This is achieved by introducing a special blank token into the target vocabulary. Optionally, identical outputs produced by multiple consecutive states may be merged and considered a single output. Because of these properties, there are groups of equivalent output sequences, which all represent the same target, as illustrated in Figure 4.1.

In standard sequence-to-sequence architectures, the value of the loss function is defined as the sum of cross entropies of the output distributions with respect to the target sequence (see Equation 2.34). In CTC, the loss is defined as the sum of cross-entropy losses of all output sequences equivalent to the given target sequence:

$$\mathcal{L}_{\text{CTC}}(\theta) = -\sum_{(x,y)\in D} \sum_{y' \sim y} \log p(y'|x, \theta) \tag{4.1}$$

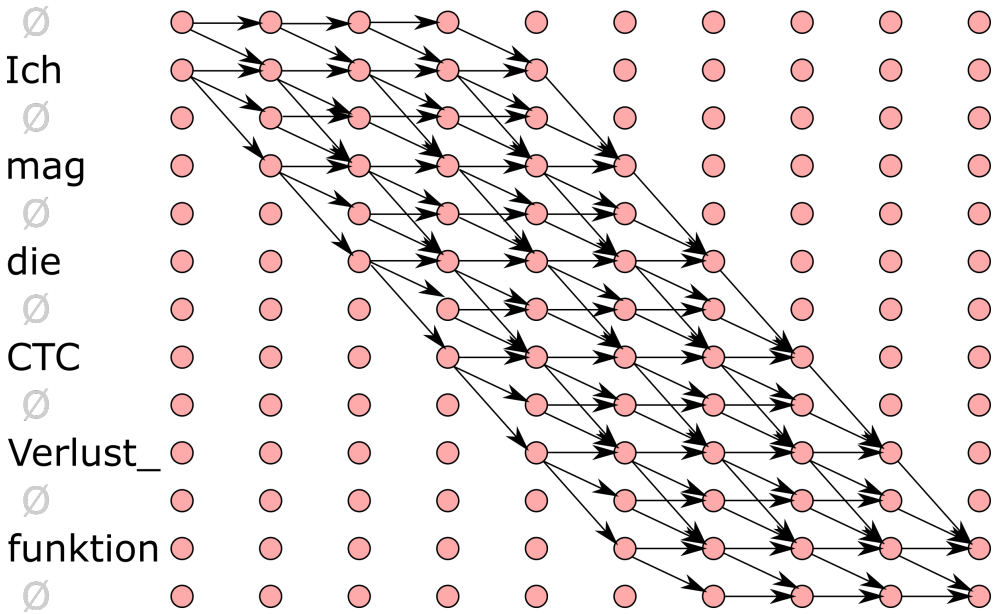



Figure 4.2: An illustration of the algorihm for the CTC loss computation. Each node denotes producing either a token from the label sequence, or the blank token. Each path from one of the two top-left nodes to one of the two bottom-right nodes corresponds to one of the equivalent sequences.

where $\sim$ denotes the equivalence relation.

The inner summation in Equation 4.1 is computed over all possible sequences equivalent to the label sequence. For technical purposes, the label sequences are limited to a fixed length, which greatly reduces the number of acceptable hypotheses. However, the number of equivalent hypotheses of a given length still grows exponentially with the sequence length – in CTC, the fixed length is always set to be longer than the label sequence.

The main difference between our approach and aligned cross-entropy (AXE) described in Section 3.2 is that in our models, we take into account all possible alignments, whereas AXE considers only the alignment with the minimum cross entropy. Moreover, we do not allow skipping positions in the ground truth.

The summation over the large set of equivalent sequences can be implemented using dynamic programming. When both the length of the output and the length of the target sequences are known, there is a constant number of blank tokens to be generated. The process of computing the loss of the whole output sequence is divided into computing the partial losses with respect to the possible label prefixes.

The CTC loss computation is illustrated in Figure 4.2. The rows represent tokens from the label sequence plus the optional blank tokens. The columns represent the output sequence. Each node in the graph denotes generating a label from an output distribution. The arrows show valid transitions between the generation steps. An

arrow can only go down one or two rows, or horizontally. The horizontal arrows denote repeated generation of the same label. These labels are later merged to form a single output. An arrow can only go two rows down when the skipped row corresponds to the blank token, so no target tokens are left out. Therefore, each path in the diagram shows one of the equivalent sequences that lead to generating the given label sequence.

Using the idea that many of the paths from left to right in the diagram share segments, we can apply dynamic programming to compute the sum of losses across all paths without the need to enumerate each of them. A node on coordinates $(i, j)$ stores the accumulated losses for all path prefixes that lead to the node, summed with the negative log-likelihood of the label on the $i$-th row being generated by the $j$-th output state. The two bottom-right nodes then store the sum of losses of all the paths.

Since the CTC loss function is differentiable with respect to the model parameters, the training of the network can be done in the standard way using the backpropagation algorithm.

## 4.2 Model Architecture

As mentioned in the previous section, training models with CTC does not impose any requirements on the model architecture. In our experiments, we aim to make a reasonable comparison between our proposed approach and the state-of-the-art autoregressive models. We adapt the Transformer model and use similar hyperparameters where applicable.

Non-autoregressive models generate the outputs in parallel, which requires that the output length is known beforehand. In autoregressive models, the end of the sequence is indicated by a special end symbol, and the constraint on maximum length is merely a technical aspect.

To maximize the ability to output empty tokens to the full extent, the output length should be set to a higher number than the length of the target sequence. Since the length estimation does not need to be accurate, we select a number $k$, and we set the target sequence length to be $k$ times longer than the source length. Note that in case the selected length is shorter than the label sequence, the model will not be able to generate the whole target sequence.

We implement the source-to-target length expansion by linear projections and state splitting. This mechanism is illustrated in Figure 4.3. After a given Transformer layer completes its computation, we linearly project the states $h_1, \ldots, h_{T_x} \in \mathbb{R}^d$ into $\mathbb{R}^{kd}$. Then, we split each of these projections into $k$ parts, which results in a $k$-times

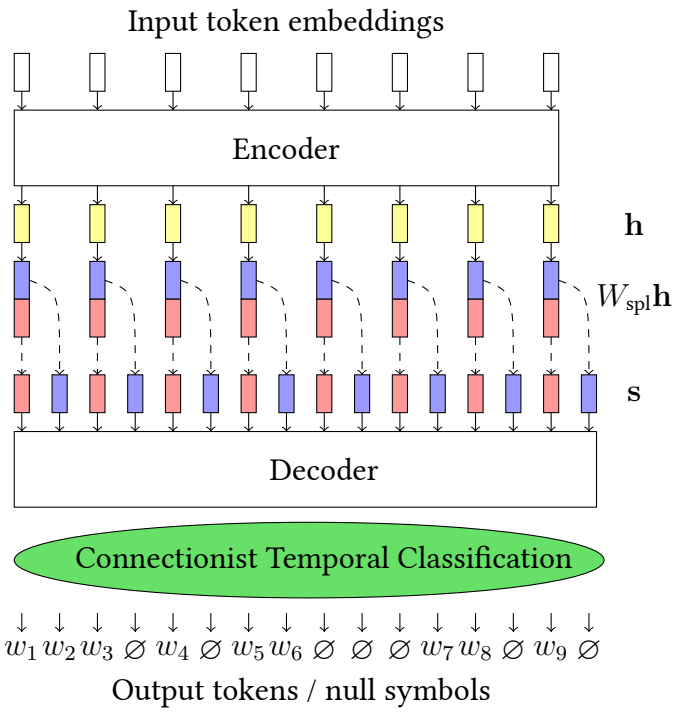



Figure 4.3: The scheme of the non-autoregressive architecture with state splitting and CTC. Image source: Libovický and Helcl (2018), Figure 1.

longer sequence of states $s_1, \ldots, s_{k \cdot T_x}$ of the original dimension $d$:

$$s_{ck+b} = \left(W_{\text{spl}} h_c + b_{\text{spl}}\right)_{bd:(b+1)d} \tag{4.2}$$

for $b = 0 \ldots k-1$ and $c = 1 \ldots T_x$ where $W_{\text{spl}} \in \mathbb{R}^{d \times kd}$ and $b_{\text{spl}} \in \mathbb{R}^{kd}$ are the trainable projection parameters.

We experiment with two placement options for the state-splitting layer. First, we try placing the state splitting at the end of the Transformer layer stack. In this scenario, there are 12 Transformer encoder layers, followed by the state-splitting layer, whose outputs are used in the output projection. In Table 4.3, this variant is called "Deep encoder". Second, we place the state-splitting layer in the middle of the Transformer layer stack, mimicking the 6-layer encoder-decoder architecture of the AR Transformer model. We use the name "Encoder-decoder" for this variant in Table 4.3. In the second variant, cross-attention can be included in the second half of the layers, which attends to the states right after state splitting. We call this model "Encoder-decoder with pos. enc." in the results table.

| | Sentence pairs | Vocabulary size |
|---|---|---|
| En – De | 4.58 M | 41,488 |
| En – Ro | 613 k | 38,440 |

Table 4.1: The sizes of the parallel training data and the wordpiece vocabularies for English–German and English–Romanian translation.

## 4.3 Preliminary Experiments

We conducted experiments with the CTC-based NAT models described above on English–German and English–Romanian translation in both directions.

**Data.** In our experiments, we use the parallel data provided by the Conference on Machine Translation (WMT) organizers. For English–German, the training data consist of the Europarl corpus (Koehn, 2005), News commentary (Tiedemann, 2012), and Common Crawl.[1] For validation, we use the WMT 13 test set (Bojar et al., 2013), and we evaluate translation quality on the WMT 14 test set (Bojar et al., 2014). For English–Romanian, the data consist of the Europarl corpus and the SETIMES corpus distributed by OPUS (Tiedemann, 2012). We use the development and test set from the WMT 16 (Bojar et al., 2016). Data sizes are shown in Table 4.1.

We preprocess the data with scripts from the `mosesdecoder` repository,[2] a part of the Moses translation toolkit (Koehn et al., 2007). Namely, we normalize the punctuation in the data, then we use the tokenizer and a truecaser. We segment the data using the wordpiece algorithm, creating a vocabulary of approximately 41k wordpieces for English–German, and 38k wordpieces for English–Romanian (Wu et al., 2016).

**Models and Training.** We implement and train our models in the Neural Monkey toolkit (Helcl and Libovický, 2017; Helcl et al., 2018). Neural Monkey is a higher-level deep learning toolkit implemented in TensorFlow (Abadi et al., 2015), aimed at fast prototyping using simple-format configuration files. We train all models for 10 epochs and select the best-scoring model based on the validation BLEU score. Training of the En–De models on a single Nvidia GeForce GTX 1080 GPU took approximately 4 weeks. Since the En–Ro parallel data was much smaller, training of the En–Ro models took 4 days.

[1] `https://commoncrawl.org/`

[2] `https://github.com/moses-smt/mosesdecoder/`

| Parameter | Value |
|---|---|
| No. of encoder layers | 6 |
| No. of decoder layers | 6 |
| Model dimension | 512 |
| Attention heads | 16 |
| Dropout probability | 0.1 |
| Feed-forward hidden size | 4,096 |
| State splitting factor | 3 |
| State splitting projection size | 1,536 |
| Vocabulary size | See Table 4.1 |
| Optimizer method | adam |
| $\beta_1$ | 0.9 |
| $\beta_2$ | 0.997 |
| $\epsilon$ | $10^{-9}$ |
| Learning rate | $10^{-4}$ |
| Fixed batch size | 20 |
| Gradient clipping | 1 |

Table 4.2: Experimental settings for the Neural Monkey experiments.

We summarize the model and training settings in Table 4.2. Note that the architecture resembles the Transformer big settings, but uses a smaller model dimension. We use the same settings for training models in both directions in both language pairs.

**Results.** Table 4.3 compares the quantitative results of our NAR models with the methods proposed by Gu et al. (2018) and Lee et al. (2018). We use Sacrebleu (Post, 2018) to compute the BLEU scores of the model outputs.

The first model of Gu et al. (2018) represents the NAT model with fine-tuning, which minimizes the KL divergence between the output distributions of the teacher and student models. In the second row, noisy parallel decoding (NPD; see Section 3.1) with 100 samples is used. Using NPD can slow down the decoding because an additional scoring step is needed using an AR model. In theory, if enough parallelism is available, this doubles the latency.

The approach of Lee et al. (2018) is iterative. We show the performance of this method with 1 iteration and with 10 iterations. Note that using more iterations slows the decoding speed of the iterative model.

| | WMT 16 | | WMT 14 | |
|---|---|---|---|---|
| | En → Ro | Ro → En | En → De | De → En |
| Gu et al. (2018) | | | | |
| Autoregressive baseline | 31.35 | 31.03 | 22.71 | 26.39 |
| NAT + FT | 27.29 | 29.06 | 17.69 | 21.47 |
| NAT + FT + NPD (100 s) | 29.79 | 31.44 | 19.17 | 23.20 |
| Lee et al. (2018) | | | | |
| Autoregressive baseline | 31.93 | 31.55 | 23.77 | 28.15 |
| 1 iteration | 24.45 | 25.73 | 13.91 | 16.77 |
| 10 iterations | 29.32 | 30.19 | 21.61 | 25.48 |
| Libovický and Helcl (2018) | | | | |
| Autoregressive baseline | 21.19 | 29.64 | 22.94 | 28.58 |
| Deep encoder | 17.33 | 22.85 | 12.21 | 12.53 |
| + weight averaging | 18.47 | 24.68 | 14.65 | 16.72 |
| + beam search | 18.70 | 25.28 | 15.19 | 17.58 |
| Encoder-decoder | 18.51 | 22.37 | 13.29 | 17.98 |
| + weight averaging | 19.54 | 24.67 | 16.56 | 18.64 |
| + beam search | 19.81 | 25.21 | 17.09 | 18.80 |
| Encoder-decoder with pos. enc. | 18.13 | 22.75 | 12.51 | 11.35 |
| + weight averaging | 19.31 | 24.21 | 17.37 | 18.07 |
| + beam search | 19.93 | 24.71 | 17.68 | 19.80 |

Table 4.3: Automatic evaluation of our CTC-based approach, compared to the two of the first non-autoregressive methods, along with autoregressive greedy-decoding baseline scores.

| | CPU | GPU |
|---|---|---|
| AR, batch=1 | 2,247 ms | 1,129 ms |
| AR, batch=100 | | 127 ms |
| NAR, batch=1 | 397 ms | 353 ms |
| NAR, batch=100 | | 41 ms |

Table 4.4: The average times to decode one sentence under different conditions.

Compared to single-pass models, the performance of the English–German CTC-based models is similar. However, the performance of the English–Romanian models is poor. This is probably due to issues with Romanian diacritics in the data, as suggested also by the poorer performance of our autoregressive baseline. Also, the enhanced techniques (NPD and iterative refiniment) outperform our method in all scenarios, but at a computational cost.

We also evaluate the effect of weight averaging and beam search (described in Section 2.4). As opposed to decoding from an AR model, beam search in CTC-based NAR models operates only on the output distributions from the single decoding step. Although in NAR models we are able to find the most likely sequence by taking the argmax of the output distribution in each position, this way we only find a single alignment. Using beam search takes into account more alignments, and the sum of the alignment probabilities can be higher than the probability of the sentence generated by argmax decoding. We see from Table 4.3 that especially weight averaging brings a substantial improvement of the translation quality.

The decoding speed for AR and NAR models under different conditions is shown in Table 4.4. We report the average time to decode a single sentence on the WMT 15 test set (Bojar et al., 2015), which consists of 2,169 sentence pairs. In the CPU setting, we are using a CPU machine with the TensorFlow session configured to use 12 CPU threads. GPU results are measured on a single Nvidia GeForce GTX 1080 GPU. We show the average latency as well as the average time to decode a sentence with batch decoding.

Figure 4.4 shows the decoding latencies on sentences from the WMT 15 test set (Bojar et al., 2015). As we can see from the relationship between the source sentence length and the decoding time, the NAR model exhibits a lower time complexity than the AR model. The latencies in the plot were measured on a CPU server using 12 threads. We use greedy decoding with a single sentence in a batch.

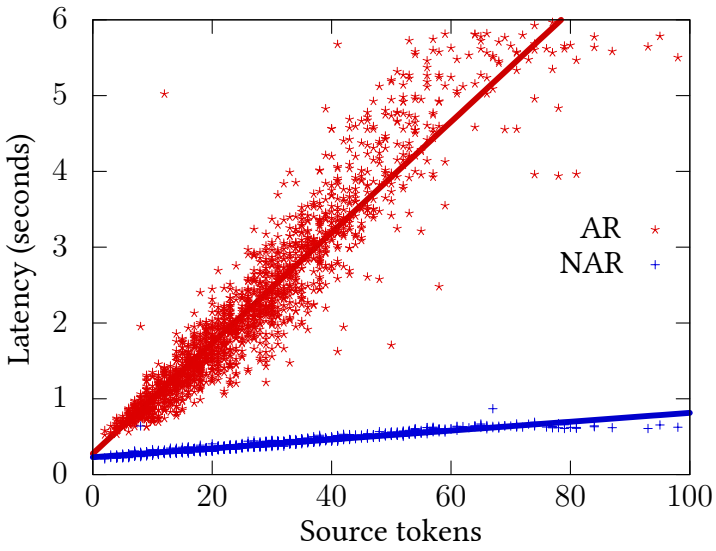



Figure 4.4: The relationship between the latency and the number of source tokens using an autoregressive and a non-autoregressive model

**Error Analysis.** We performed manual error analysis of the model outputs. We sampled 100 random sentences from the English–German test set for each translation direction. For each sentence in the test set, we first classified whether the sentence was correctly translated or whether it was at least comprehensible.

We then identified four types of errors which were prevalent the most in the outputs: 1) the output sentence was too short, 2) a verb was missing in the output, 3) a named entity that consists of multiple subwords was not properly constructed, and 4) another multi-subword word was corrupted.

The prevalence of the different types of errors that we found is shown in Table 4.5. We observed that when the outputs of the AR model were incorrect, this was further amplified in the NAR model, rendering many sentences incomprehensible. Overall, we can see that the number of completely correct translations in the sample was low for the NAR model, less than a quarter of the sentences in both directions. One of the most common errors in the English → German direction was the omission of a verb at the end of the sentence. This is likely a manifest of the multimodality problem – in German, there are two ways of composing the past tense, one of which inflects the verb in place and the other puts the verb to the end of the sentence. In both directions, we noticed many errors connected to words which are represented with multiple wordpieces, such as rare or infrequent words or named entities.

Many of the errors are related to the fluency of the output, which might be caused by the weakened language model in the NAR models. We address this issue by incorporating an external language model into the decoding process, and describe these experiments in the following section.

| | En → De | | De → En | |
|---|---|---|---|---|
| | AR | NAR | AR | NAR |
| Correct | 65 | 23 | 67 | 13 |
| Comprehensible | 93 | 71 | 92 | 51 |
| Too short | 1 | 16 | 0 | 36 |
| Missing verb | 4 | 35 | 0 | 8 |
| Corrupt. named entity | 1 | 27 | 8 | 21 |
| Corrupt. other words | 1 | 20 | 0 | 46 |

Table 4.5: Results of a cursory manual analysis of the most frequent types of errors in English ↔ German translation models.

## 4.4 Improving Fluency with n-gram Language Models

*This section is based on the paper* "Improving Fluency of Non-Autoregressive Neural Machine Translation"*, joint work with Zdeněk Kasner and Jindřich Libovický, published online at* `arXiv.org`*.*

In this section, we describe our effort to improve the language modeling capabilities of the CTC-based NAT model. As described in the previous section, frequent errors that non-autoregressive models make include missing verbs or bad composition of words from multiple subwords. The literature on NAR models also points out that token repetition is also a frequent issue, although in alignment-based models (see Section 3.2), the repeated tokens are often merged together. (The models can still generate repeated words; for example, by interleaving them with the blank token.)

Language models (LMs) have been a very successful component in the classical phrase-based statistical MT models (Koehn, 2009). However, in neural MT, language modeling is handled by the decoder part, which can be viewed as a LM conditioned on the encoder hidden representation of the source sentence.

By imposing the conditional independence assumption in non-autoregressive models (Equation 3.1) we impair the implicit language model, which leads to the lower fluency of the translations.

In this study, we propose to enhance the beam search algorithm (see Section 2.4) by a number of fluency-oriented features, which we include in the hypothesis scoring model:

$$\text{score}(y|x) = \log p(y|x) + \sum i = 1^n w_i \cdot \Phi_i(y) \tag{4.3}$$

| | Data size |
|---|---|
| Parallel | |
| En – De | 4.58 M |
| En – Ro | 613 k |
| Monolingual | |
| En | 20 M |
| De | 20 M |
| Ro | 2.2 M |

Table 4.6: The sizes of the parallel and monolingual data used for training.

where $p(y|x)$ is the probability given by the CTC model, $\Phi_i(y)$ are the features of the scoring model, and $w_i$ are their respective weights. This rescoring approach is somewhat similar to the NPD proposed by Gu et al. (2018), in which they create more hypotheses by sampling from the fertility model and rescore them using an AR Transformer model as a scorer.

We experiment with the following three features in the scoring model. First, we use the score of $y$ from an n-gram language model. We use KenLM, an efficient implementation of n-gram LMs in C++ (Heafield, 2011). Second, we use the ratio of the blank tokens in the output alignment $y$:

$$\Phi_{\text{blank-ratio}}(y) = \max(0, \frac{B}{T_y - B} - 4) \tag{4.4}$$

where $B$ is the number of blank tokens in the output, $T_y - B$ is the number of non-blank tokens. Third, we count the number of trailing blank symbols in the hypothesis:

$$\Phi_{\text{trailing-blanks}}(y) = \max(0, B^T - T_x) \tag{4.5}$$

where $B^T$ is the number of blank tokens at the end of the sequence, and $T_x$ is the length of the source sentence.

### 4.4.1 Experiments and Results

We follow an experimental setup similar to that in the previous section with a few differences. In addition to the parallel data, we use backtranslated monolingual data (dataset sizes are shown in Table 4.6), which we mix with upsampled parallel data so that the ratio of the synthetic and parallel data in the training dataset is approximately 1:1. We train the AR baselines and the NAT model on this dataset, so that the models with the LM scorer have access to the same data. Instead of the word-piece vocabulary, we use SentencePiece (Kudo and Richardson, 2018) with 50k tokens

| | WMT 16 | | WMT 14 | | Latency (ms) |
|---|---|---|---|---|---|
| | En → Ro | Ro → En | En → De | De → En | |
| Baseline | | | | | |
| AR, greedy | 25.89 | 33.54 | 27.29 | 31.06 | 1,664 |
| AR, beam 5 | 26.46 | 34.06 | 27.71 | 31.85 | 3,848 |
| NAR without LM | 19.88 | 28.99 | 19.55 | 23.04 | 233 |
| LM scoring | | | | | |
| Beam 1 | 19.74 | 29.65 | 20.59 | 24.11 | 337 |
| Beam 5 | 22.46 | 33.01 | 23.61 | 27.19 | 408 |
| Beam 10 | 23.33 | 33.29 | 24.27 | 27.83 | 526 |
| Beam 20 | 24.11 | 33.51 | 24.41 | 28.14 | 1,097 |

Table 4.7: The model performance (in terms of BLEU score and latency) for the English–Romanian and English–German translation models with LM scoring with various beam sizes.

in the vocabulary. We train a 5-gram language model on the monolingual data using KenLM.[3] We use structured perceptron (Huang et al., 2012) to learn the feature weights in the rescoring model. We train the perceptron on the development set, using half of the dataset for training and the other half for testing.

Table 4.7 shows the BLEU scores on the test data and the latencies of the NAR models using various beam sizes for decoding. We can see that even for low beam size settings, the beam rescoring model achieves considerable improvements over the NAR baseline. The rescoring model does affect the decoding speed as the beam size grows, but the translation improvement gained by beam sizes up to 10 does not cost too much in terms of speed.

We conducted an ablation study of the features in the rescoring model. The BLEU scores of the rescoring model with different sets of rescoring features (LM, blank vs. non-blank symbol ratio, and the number of trailing blank tokens) for various beam sizes are shown in Table 4.8. In the first row, only the CTC score is used (i.e., no rescoring is performed). In the second row, the LM score is added to the model. We see that the addition of the LM alone accounts for most of the performance gain. In the third row, we add the blank vs. non-blank symbol ratio to the CTC and LM scores. The fourth row shows the results of the scoring model with the full set of features.[4]

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

[4] Note that we use a different test set (WMT 15 news test set). For comparison with the other models on this test set, we refer the reader to Kasner et al. (2020) or Kasner (2020).

| Beam Size | 1 | 5 | 10 | 20 |
|---|---|---|---|---|
| CTC score | 21.67 | 22.06 | 22.13 | 22.17 |
| + LM | 22.05 | 24.64 | 24.77 | 25.12 |
| + blank symbol ratio | 22.21 | 24.92 | 25.12 | 25.35 |
| + trailing blank symbols | 22.68 | 25.50 | 25.93 | 26.03 |

Table 4.8: The translation score of the English–German NAT model when using various combinations of features and beam sizes measured on the WMT 15 test set.

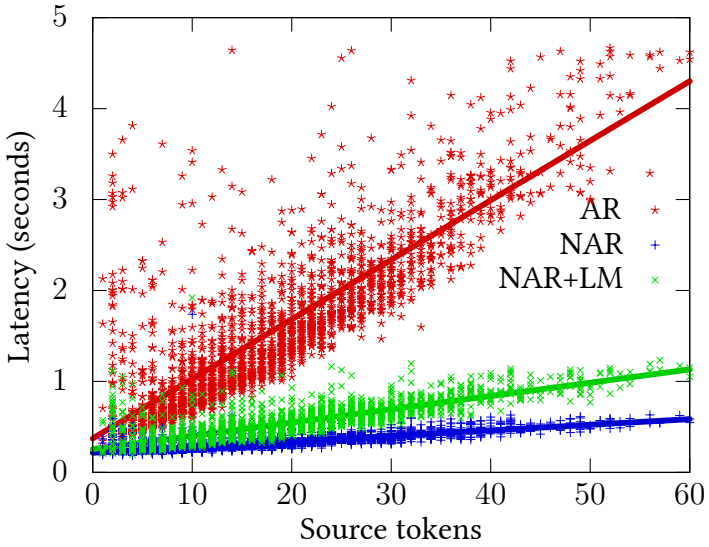



Figure 4.5: The relationship between the latency and the number of source tokens in AR, NAR, and NAR models with LM.

Similarly to Figure 4.4 in the previous section, we analyze the relationship between the latencies in models with and without LM rescoring in Figure 4.5. We can see that even though the latency of the NAR+LM system grows with the length of the source sentence slightly faster than the latency of the pure NAR system, it is still much faster than the AR counterpart.

To illustrate how beam rescoring can improve the fluency of the outputs of the NAR model, we handpick a few examples from the German–English test set and show them in Table 4.9.

| | |
|---|---|
| Source | Die Armee teilte zugleich mit, dass in den vergangenen 24 Stunden sieben Soldaten getötet worden seien. |
| Reference | The army announced at the same time that over the past 24 hours, seven soldiers have been killed. |
| AR | The army also announced that seven soldiers have been killed over the past 24 hours. |
| NAR | The army also announced that seven soldiers had killed killed in the last 24 24 hours.<br>*Missing verb, Repeated tokens* |
| NAR + LM | The army also announced that the seven soldiers had been killed in the last 24 hours.<br>*Fixed issues but inserted a definite article* |
| Source | Wir waren früher ein freundliches Land, wie kann es sein, dass wir nun kämpfen? |
| Reference | We used to be one friendly country in the past, how did it happen that we are fighting now? |
| AR | We used to be a friendly country, how can we fight now?<br>*Translation too short* |
| NAR | We were a friendly country, how can it we are fighting now?<br>*Missing words* |
| NAR + LM | We were a friendly country, how can it be that we are fighting now?<br>*Partially correct* |
| Source | Seiner Ansicht nach könnten alle Mitglieder beider Vereine künftig wieder an einem Strang ziehen. |
| Reference | In his opinion, all members of each club could come together again in the future. |
| AR | In his view, all members of both clubs could pull together again in the future. |
| NAR | In believes that all members of both clubs could pull together future.<br>*Ungrammatical beginning and end* |
| NAR + LM | He believes that all the members of both clubs could pull together.<br>*Fluency corrections; inserted "the" again, this time not a mistake* |

Table 4.9: Handpicked examples of the model outputs for German–English translation.

## 4.5 Limitations

The proposed approaches described in this chapter and their experimental evaluation have certain limitations, related to the discussion in Section 3.6. We list the most relevant drawbacks and we attempt to address them in the experiments presented in the next chapter.

One of the main weaknesses of the preliminary experiments is the relatively low translation quality of both the autoregressive baseline and the CTC-based models compared to the other approaches. The main cause of the poor performance of the NAT model is that the models were trained without knowledge distillation. The causes of the poor translation quality of the AR baseline models include not using backtranslation (see the AR results without and with backtranslation in Tables 4.3 and 4.7 in Sections 4.3 and 4.4 respectively) and using a smaller and perhaps noisier dataset than what can be obtained.

Another weakness of the models from the previous sections is the relatively low decoding speed in absolute numbers. Although we achieve reasonable speed-up compared to an AR baseline, the latency can be substantially lower than we measured, both on CPU and on a GPU.

# 5 Experiments

As we discussed in Section 3.6, the flaws in the evaluation methodology adopted by current research of non-autoregressive neural machine translation (NAT) models make it difficult to identify approaches suitable in production-ready conditions. In this chapter, we present further experiments with NAT models trained with connectionist temporal classification (CTC). We focus on a fair comparison with other non-autoregressive approaches as well as state-of-the-art optimized autoregressive methods. In line with our previous experiments (see Chapter 4) and also with the rest of the related work, we conduct experiments on English-German translation.

As previously stated (see Section 3.1), knowledge distillation (Kim and Rush, 2016) is a crucial element that brings the performance of non-autoregressive (NAR) methods closer to that of autoregressive (AR) methods. We use strong AR teachers as both our baseline models and as the source of the artificial target side data for training of the distilled models. We describe our teacher models in detail in Section 5.1

In Section 5.2, we describe the training of the NAR student models. We follow the approach described in Chapter 4. Key changes to the CTC-based models include training the student models on distilled data and using a faster implementation of the translation system.

In the analysis of the results presented in Section 5.3, we try to connect the separate worlds of NAT research and the bids to improve the efficiency of translation models. Our student models outperform all of the NAR models in terms of BLEU on the standard WMT 14 translation benchmark. However, at the same time, the models score relatively poorly in comparison to the current state-of-the-art AR translation models, which suggests that the gap in translation quality between AR and NAR models is still a problem, contrary to some of the claims in the literature (Gu and

Kong, 2021; Saharia et al., 2020). We also present an extensive analysis of the decoding speed under different conditions and conclude that without future advancements in this field, optimized AR models will continue to achieve superior results over NAR models.

## 5.1 Autoregressive Teacher Models

*This section is based on the paper* "The University of Edinburgh's English-German and English-Hausa Submissions to the WMT21 News Translation Task"*, joint work with Pinzen Chen, Ulrich Germann, Laurie Burchell, Nikolay Bogoychev, Antonio Valerio Miceli Barone, Jonas Waldendorf, Alexandra Birch, and Kenneth Heafield, published at WMT 2021.*

In this section, we describe the experimental settings for training the AR teacher models, which we use for knowledge distillation to prepare training data for our NAR student models.

In our English-German experiments, we use autoregressive models from our submission to the Conference on Machine Translation (WMT) 2021 News Translation Shared Task (Chen et al., 2021). We use these models both as strong AR baselines and as teacher models for generating distilled data for the NAR student models, described in Section 5.2. The models were trained on cleaned parallel data augmented with backtranslated monolingual data, using Marian, a C++ toolkit for training and decoding from neural machine translation (NMT) models (Junczys-Dowmunt et al., 2018).

**Data Cleaning.** We prepare the training dataset consisting of the following parts: First, we use clean parallel data from the Europarl corpus (Koehn, 2005), the Tilde MODEL – RAPID corpus (Rozis and Skadiņš, 2017), and the News Commentary corpus from OPUS (Tiedemann, 2012). Next, we include sources of crawled parallel data from the web, which are considered noisy. These include Paracrawl (Esplà et al., 2019), Common Crawl[1], WikiMatrix (Schwenk et al., 2019), and the Wikipedia Parallel Titles Corpus[2]. Finally, we use backtranslation (Sennrich et al., 2016a) of monolingual data obtained from News Crawl. We train our own models to generate the backtranslations on cleaned parallel data, as described below.

We perform the following filtering techniques on the gathered data (both clean and noisy) to improve the overall quality of the parallel data.

[1] `https://commoncrawl.org/`

[2] `https://linguatools.org/tools/corpora/wikipedia-parallel-titles-corpora/`

| Data source | Raw size | | Size after cleaning |
|---|---|---|---|
| Europarl | 1.83 | | |
| RAPID | 1.63 | 3.86 | 3.1 |
| News Commentary | 0.40 | | |
| Paracrawl | 82.64 | | |
| WikiMatrix | 5.47 | 91.98 | 84.6 |
| Common Crawl | 2.40 | | |
| Wikititles | 1.47 | | |
| total | | 95.84 | 87.7 |

Table 5.1: The sizes of the individual raw corpora before and after rule-based cleaning, in millions of sentence pairs.

We start with deterministic rule-based filtering[3] and deduplication. We remove all sentence pairs containing non-printing characters, empty sentences, and sentences longer than 120 words; we also remove all sentence pairs with length ratio of less than 0.6 (0.4 for Wikititles), sentences in which over 40% of characters do not constitute tokens, and sentences in which more than 50% were non-alphabetic characters. We run language identification using fastText (Joulin et al., 2017, 2016) and remove all sentence pairs classified as not English-German.

The data sizes before and after the rule-based filtering are shown in Table 5.1. Note that the vast majority of the training corpus consists of data from noisy sources.

To further clean the data, we apply dual cross-entropy filtering, as proposed by Junczys-Dowmunt (2018) and described in the data cleaning paragraph in Section 2.3.1. We train two Transformer base models (one for each translation direction) for dual cross-entropy filtering using the clean part of the data after the rule-based cleaning step.

We score the crawled part of the parallel data using the trained models and we sort the sentence pairs according to the score. To estimate how many of the crawled sentence pairs can we consider clean, we train the big variant of the Transformer translation models in both directions on different amounts of the scored data (following the teacher model hyperparameter settings in Table 5.4). We use 25%, 50%, 75% and 100% of the data, taking the highest-scoring sentence pairs for training. Based on the BLEU scores achieved by the Transformer big models on the development data (the WMT 2019 test set, Barrault et al., 2019), we declare the 75% as the "clean" portion of the crawled data. The develoment scores achieved by the trained models are shown in Table 5.2.

[3] https://github.com/browsermt/students/blob/master/train-student/clean/

| Percentage | 25% | 50% | 75% | 100% |
|---|---|---|---|---|
| No. of sentences (M) | 21.9 | 43.9 | 66.5 | 87.7 |
| En → De | | 43.68 | 43.40 | 42.70 |
| De → En | 41.47 | 41.64 | 42.15 | 42.02 |

Table 5.2: Development BLEU scores of the Transformer models trained on different amounts of scored crawled parallel data in both directions.

| Data source | | Raw size | Size after cleaning |
|---|---|---|---|
| English News Crawl | 2018 | 18.11 | 17.90 |
| | 2019 | 33.60 | 32.80 |
| | 2020 | 41.43 | 40.33 |
| | total | 93.14 | 91.03 |
| German News Crawl | 2018 | 38.65 | 37.42 |
| | 2019 | 57.62 | 56.33 |
| | 2020 | 53.67 | 52.46 |
| | total | 149.94 | 146.22 |

Table 5.3: The sizes of the monolingual data (in millions of sentences) used for the training, including rule-based data filtering

**Backtranslation.** We train three additional translation models (four in total with the one from the data cleaning step) on the filtered parallel dataset to create back-translations (Sennrich et al., 2016a). We use the same hyperparameter settings as for the teacher models (see Table 5.4), except for different random seeds for parameter initialization.

We translate the monolingual data using the four models in an ensemble. We limit the target sentence length to 150 tokens and use beam search decoding with 6 hypotheses in the beam, with the length normalization parameter set to 1.

As the source of the monolingual data, we use the News Crawl datasets from years 2018, 2019, and 2020, as released by the WMT organizers (Bojar et al., 2018; Barrault et al., 2019, 2020). In total, we gathered 91 million English sentences for backtranslation into German and 146 million German sentences for backtranslation into English. Table 5.3 shows the sizes of the monolingual data before and after applying the rule-based filtering described in the paragraphs above.

We follow the approach of Caswell et al. (2019) and we tag the backtranslated sentences with a special token on a first position, as explained in Section 2.3.1.

| Parameter | Marian config variable | Value |
|---|---|---|
| No. of encoder layers | `enc-depth` | 6 |
| No. of decoder layers | `dec-depth` | 6 |
| Model dimension | `dim-emb` | 1,024 |
| Feed-forward state dimension | `transformer-dim-ffn` | 4,096 |
| Attention heads | `transformer-heads` | 16 |
| Vocabulary size | | 32,000 |
| Optimizer method | `optimizer` | adam |
| $\beta_1$ | | 0.9 |
| $\beta_2$ | `optimizer-params` | 0.998 |
| $\epsilon$ | | $10^{-9}$ |
| No. of batches per update | `optimizer-delay` | 2 |
| Fit batch to available memory | `mini-batch-fit` | true |
| Learning rate | `learn-rate` | $10^{-4}$ |
| Learning rate warmup | `lr-warmup` | 8,000 |
| Learning rate decay | `lr-decay-inv-sqrt` | 8,000 |
| Gradient clipping[4] | `clip-norm` | 0 |

Table 5.4: The hyperparameters of the teacher models. The same values were used for training the models for backtranslation.

**Teacher Model Training.** We train the teacher models on shuffled concatenation of the authentic parallel and tagged backtranslated data. As with the backtranslation models, we train four models with different seeds for random initialization in each direction. We show the hyperparameter values in Table 5.4.

After the training on mixed parallel and backtranslated data converged, we continued the training of the models on parallel data only.

**Knowledge Distillation.** The teacher models are used to create artificial targets for the student models (Kim and Rush, 2016). For each translation direction, we translate the source side of the parallel data and the authentic monolingual data (in the source language) using the ensemble of the teacher models. We do not create wholly synthetic datasets by forward-translating backtranslated data.

For generating the knowledge-distilled dataset, we use beam search with the beam size of 12 and the normalization parameter set to 0.8. Similarly to generating the backtranslated data, we limit the target sentence length to 150 tokens.

[4]We did not use gradient clipping because of issues in the Marian toolkit implementation.

| Model | Base architecture | Encoder layers | Decoder layers |
|---|---|---|---|
| Teacher | Transformer big | 6 | 6 |
| Large | Transformer big | 6 | 6 |
| Base | Transformer base | 6 | 6 |
| Small | | 3 | 3 |
| Micro | | 2 | 2 |
| Tiny | | 1 | 1 |

Table 5.5: The hyperparameters of the student models.

## 5.2 Student Models

In this section, we describe our non-autoregressive student models. We implemented the CTC-based Transformer architecture in the Marian toolkit (Junczys-Dowmunt et al., 2018). For the computation of the CTC loss, we use the warp-ctc library,[5] an efficient parallelized implementation for GPUs (Amodei et al., 2016).

We experiment with different hyperparameter settings that control the size of the student model. Our baseline non-autoregressive model is a big Transformer model with 6 encoder layers, followed by the state-splitting layer, and another 6 decoder layers. Apart from the CTC-specific configuration, we use the same hyperparameters as in the teacher models, shown in Table 5.4. In addition, we train four smaller models based on the Transformer base hyperparameters – using model dimension of 512, feed-forward state dimension of 2,048, and 8 attention heads. Each smaller model uses a different number of encoder and decoder layers. We show the number of encoder and decoder layers in Table 5.5. In all settings, we use the splitting factor of 3 in the state splitting layer.

All models were trained for approximately three weeks on four Nvidia Tesla P100 GPUs. Every 8,000 updates of the training, a model checkpoint was saved and validated. As the validation dataset, we used the concatenation of three WMT test sets, from years 2018, 2019, and 2020 (Bojar et al., 2018; Barrault et al., 2019, 2020). Figures 5.1 and 5.2 show the training progress of the student models. Each plot shows the validation BLEU scores after a given number of parameter updates.

[5] https://github.com/baidu-research/warp-ctc/

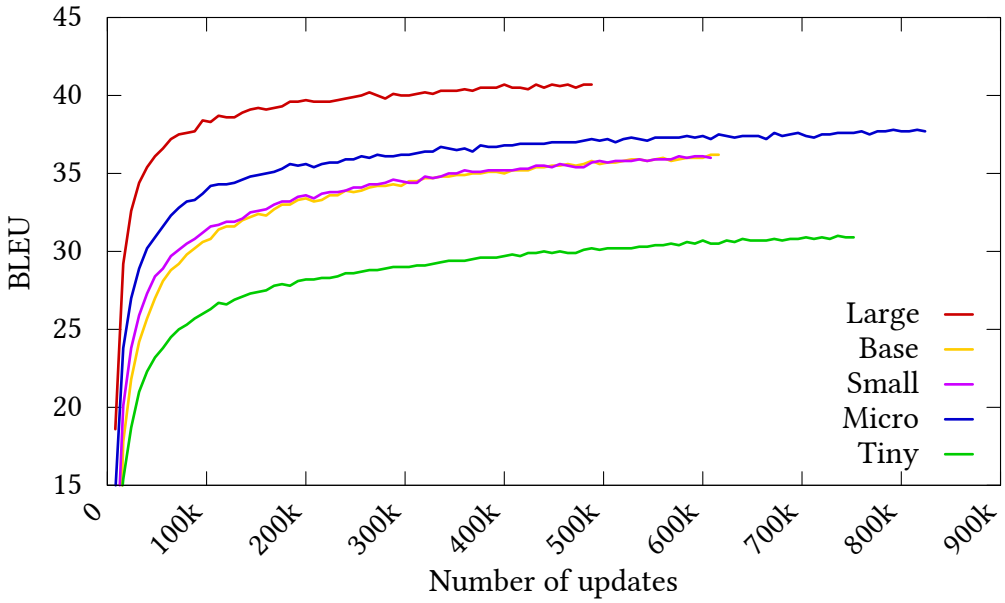



Figure 5.1: The learning curves of the English → German NAT models. The BLEU scores are reported on the concatenation of the test sets from WMT 18, 19, and 20.

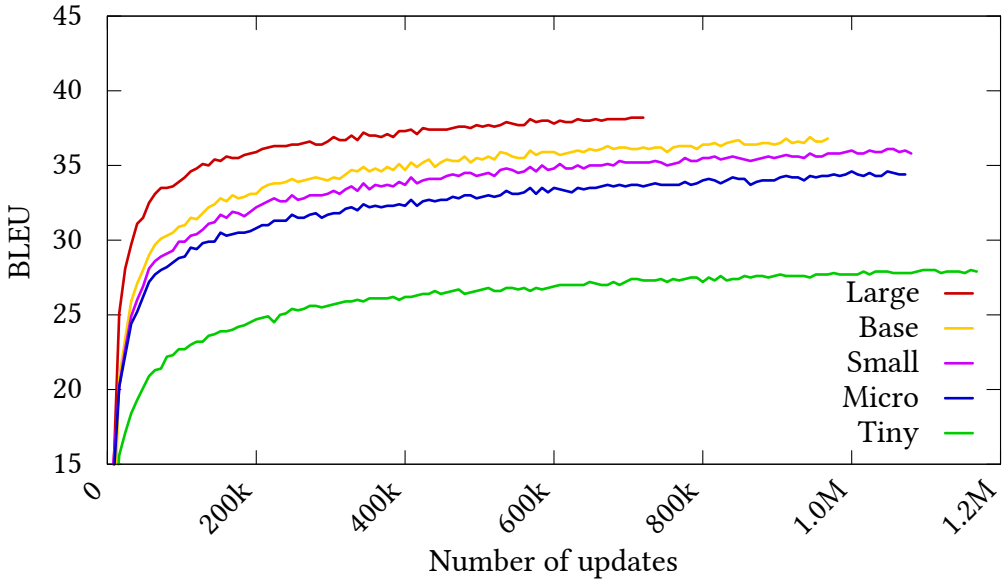



Figure 5.2: The learning curves of the German → English NAT models. The BLEU scores are reported on the concatenation of the test sets from WMT 18, 19, and 20.

### 5.2.1 Lexical Shortlist

Shortlisting is a method used for speeding up the decoding out of a translation model, originally proposed by Jean et al. (2015) as means to tackle the open vocabulary problem on the generation side (see Section 2.1.1). Since then, it has been used as an optimization method for creating efficient translation systems (Kim et al., 2019; Bogoychev et al., 2020). In our experiments, we employ shortlisting as one of the standard optimization techniques to find out whether it has a positive effect on the decoding speed not only in AR models, but also in NAR models.

A lexical shortlist limits the number of possible output tokens, reducing the dimension of the output projection matrix (recall the dimension of the output matrix is otherwise $d \times |V|$) and thus saving time on matrix multiplication operation. Output probabilities of the tokens which are not in the shortlist are not computed (and these tokens are never generated).

To use lexical shortlists during decoding, we need to create a bilingual lexicon first. Using a parallel corpus, we compute the word alignment. Based on the alignment, the bilingual lexicon is the set of target tokens aligned with each source token, along with the occurrence frequencies.

During decoding, a new lexical shortlist is generated for every new batch. To create a shortlist for a given batch, the potential translations of every source token are added to the shortlist. Specifically in our settings, we include 100 most frequent and 100 most probable translations of each source token.

## 5.3 Results

In this section we analyze the results both in terms of translation quality in Section 5.3.1 and decoding speed in Section 5.3.2. We discuss the results from the perspectives of the related literature on the NAT models and the submissions to the Efficiency Shared Task in Section 5.3.3.

### 5.3.1 Translation Quality

We compare the results of our models to both related work on NAT and to the results of the WMT 21 Efficiency Shared Task which features highly optimized AR translation models (Heafield et al., 2021). In the NAR literature, the WMT 14 test set (Bojar et al., 2014) is used as a standard benchmark. On the other hand, the efficiency task uses the recent test set from WMT 21 (Akhbardeh et al., 2021). In this section, we present the results on both datasets.

**Results on the WMT 21 test set.** As argued in Section 2.5, BLEU alone is not necessarily a reliable automatic metric. In line with the evaluation methodology of the efficiency task, we also measure COMET (Rei et al., 2020) and ChrF score (Popović, 2015). We do not perform human evaluation.

Table 5.6 shows the results of automatic evaluation on the WMT 21 news translation test set. In the English → German direction, the test set consists of 1,002 sentences along with three different reference translations. In the German → English direction, there are 1,000 sentences with two reference translations each. We measure multi-reference BLEU score using Sacrebleu (Post, 2018) and we report confidence intervals computed with bootstrap resampling. We report the Sacrebleu signatures for English → German[6] and German → English[7] directions. We compute the ChrF score separately on each reference translation set using Sacrebleu, and we report the average.[8] Finally, we compute the COMET scores with the `wmt20-comet-da` model of the COMET version `dd2298` (1.0.0rc9).

We observe that in both translation directions, the AR models outperform the NAR models. The performance gap between the models grows further with beam search and ensembling. We can also see that knowledge distillation has a positive effect on both AR and NAR models, with the student AR model matching the performance of the ensemble of four large teacher models. We also note that the difference in the COMET score is bigger than in BLEU, which might suggest that NAR models would rank poorly in human evaluation, despite achieving reasonable BLEU scores.

The results on the test set confirm the ranking of the NAR models as seen during training (see Figures 5.1 and 5.2), including the interesting exception of the English → German Micro model. Otherwise, the larger the student model is, the better scores it achieves.

Finally, we see that using a lexical shortlist does not have an effect on the translation quality in all model variants. However, we see that shortlisting impairs the performance of the Transformer base model when decoding with beam search.

**Results on the WMT 14 test set.** We present automatic evaluation results measured on the WMT 14 test set to provide a comparison to the related work on non-autoregressive models (see Table 3.1 in Section 3.6 for reference). In Table 5.7 we show the BLEU scores achieved by our NAR student models on this test set.

Since many of the related approaches stop the training early after 300 thousand updates (Gu et al., 2018; Gu and Kong, 2021), we report the scores of our models both at this point in training, and after the training reached convergence.

[6] En→De: `nrefs:3|bs:1000|seed:12345|case:mixed|eff:no|tok:13a|smooth:exp|version:2.0.0`
[7] De→En: `nrefs:2|bs:1000|seed:12345|case:mixed|eff:no|tok:13a|smooth:exp|version:2.0.0`
[8] ChrF; En↔De: `nrefs:1|case:mixed|eff:yes|nc:6|nw:0|space:no|version:2.0.0`

| **En → De** | Full output projection | | | Shortlist | | |
|---|---|---|---|---|---|---|
| | ChrF | COMET | BLEU | ChrF | COMET | BLEU |
| Single greedy AR | | | | | | |
| Large | 59.2 | 0.4110 | 50.5 ±1.3 | 59.2 | 0.4124 | 50.6 ±1.3 |
| Base | 58.1 | 0.3881 | 47.9 ±1.3 | 58.1 | 0.3875 | 47.9 ±1.2 |
| Single beam AR | | | | | | |
| Large | 58.8 | 0.4053 | 50.8 ±1.3 | 58.8 | 0.4144 | 47.9 ±1.2 |
| Base | 57.9 | 0.3873 | 48.0 ±1.3 | 55.1 | 0.2666 | 39.3 ±1.1 |
| Ensemble beam AR | | | | | | |
| Large | 59.5 | 0.4332 | 52.2 ±1.3 | 59.4 | 0.4303 | 52.2 ±1.3 |
| Student AR | | | | | | |
| Base | 59.5 | 0.4550 | 51.6 ±1.2 | 59.6 | 0.4564 | 51.6 ±1.2 |
| NAR models | | | | | | |
| Large | 58.6 | 0.1485 | 47.8 ±1.2 | 58.7 | 0.1442 | 47.7 ±1.2 |
| Base | 56.3 | -0.0521 | 41.8 ±1.1 | 56.3 | -0.0545 | 41.8 ±1.1 |
| Small | 56.2 | -0.0752 | 41.9 ±1.1 | 56.2 | -0.0773 | 41.9 ±1.2 |
| Micro | 57.3 | -0.0083 | 43.5 ±1.1 | 57.4 | -0.0085 | 43.6 ±1.1 |
| Tiny | 53.6 | -0.3333 | 34.7 ±1.0 | 53.8 | -0.3346 | 34.8 ±1.0 |

| **De → En** | Full output projection | | | Shortlist | | |
|---|---|---|---|---|---|---|
| | ChrF | COMET | BLEU | ChrF | COMET | BLEU |
| Single greedy AR | | | | | | |
| Large | 61.9 | 0.5868 | 48.4 ±1.3 | 61.9 | 0.5866 | 48.5 ±1.3 |
| Base | 61.0 | 0.5532 | 47.0 ±1.3 | 60.5 | 0.5534 | 47.1 ±1.3 |
| Single beam AR | | | | | | |
| Large | 61.5 | 0.5885 | 49.2 ±1.2 | 61.2 | 0.5659 | 43.9 ±1.1 |
| Base | 60.7 | 0.5534 | 47.4 ±1.3 | 58.0 | 0.4591 | 38.5 ±1.2 |
| Ensemble beam AR | | | | | | |
| Large | 62.0 | 0.5954 | 50.6 ±1.3 | 62.3 | 0.5970 | 50.8 ±1.3 |
| Student AR | | | | | | |
| Base | 63.3 | 0.6115 | 51.1 ±1.3 | 63.3 | 0.6112 | 51.1 ±1.3 |
| NAR models | | | | | | |
| Large | 61.6 | 0.3296 | 46.4 ±1.4 | 61.6 | 0.3288 | 46.4 ±1.4 |
| Base | 61.4 | 0.2957 | 45.8 ±1.3 | 61.4 | 0.2663 | 45.7 ±1.3 |
| Small | 61.0 | 0.2462 | 44.6 ±1.3 | 61.0 | 0.2454 | 44.6 ±1.3 |
| Micro | 59.6 | 0.1475 | 42.3 ±1.4 | 59.6 | 0.1468 | 42.3 ±1.4 |
| Tiny | 55.9 | -0.1558 | 34.4 ±1.3 | 55.9 | -0.1560 | 34.4 ±1.3 |

Table 5.6: Quantitative results of the German ↔ English translation models on the WMT 21 test set using ChrF, COMET, and BLEU.

| | En → De | | De → En | |
|---|---|---|---|---|
| Model | 300k | Final | 300k | Final |
| Large | 27.6 | 28.1 | 29.5 | 30.9 |
| Base | 22.4 | 23.7 | 27.7 | 29.8 |
| Small | 22.4 | 23.7 | 26.5 | 28.7 |
| Micro | 23.6 | 25.1 | 25.5 | 27.2 |
| Tiny | 19.0 | 20.3 | 19.5 | 21.5 |

Table 5.7: The BLEU scores of the *single best* models on the WMT 14 test set after 300k updates and at the end of the training.

| | En → De | | De → En | |
|---|---|---|---|---|
| Model | 300k | Final | 300k | Final |
| Large | 27.7 | 28.4 | 30.0 | 31.3 |
| Base | 22.4 | 23.7 | 28.1 | 30.3 |
| Small | 22.5 | 23.6 | 26.7 | 29.1 |
| Micro | 23.7 | 25.0 | 25.1 | 27.5 |
| Tiny | 19.0 | 20.3 | 19.6 | 21.7 |

Table 5.8: The BLEU scores of the *averaged* models on the WMT 14 test set after 300k updates and at the end of the training.

Table 5.8 shows the WMT 14 BLEU scores with checkpoint averaging. In each variant, we take the average parameters of the five best scoring models as measured on the validation set (either before the 300,000th update or overall). In contrast to our previous experiments in Chapter 4 (see Table 4.3), we found that checkpoint averaging only has a small effect on the translation quality in terms of BLEU.

### 5.3.2 Decoding Time

In this section, we analyze the decoding speed of the English → German NAR models. We aim at recreating the evaluation conditions following the WMT 21 Efficiency Shared Task (Heafield et al., 2021). We measure the decoding latency and throughput in different hardware environments.

The decoding time is measured on a dataset containing one million sentences to minimize the effect of the model loading overhead.

For measuring the CPU times, we use the same environment as the WMT 21 shared task organizers, which is a dual-socket Intel Xeon Gold 6354 from Oracle Cloud, a 36-core CPU server. For GPU efficiency, we use an Nvidia A100 GPU. We also include our results on an Nvidia P100 GPU which has been used often in the literature (see Table 3.2 in Section 3.6).

**GPU Latency and Throughput.** Figures 5.3 and 5.4 plot the times to translate the test set on P100 and A100 GPUs, respectively. Each figure shows the relationship between the batch size and the decoding time in seconds and includes a table with the measured data. The runs corresponding to the missing entries in the table did not finish in 24 hours.

Setting aside the absolute values (as expected, an A100 is faster than a P100), both GPU settings yield similar results. For AR models, increasing the batch size heavily reduces the overall decoding speed and eventually surpasses the large NAR models. Increasing the batch size speeds up the decoding in NAR models as well, but the effect is diminished for larger batch sizes.

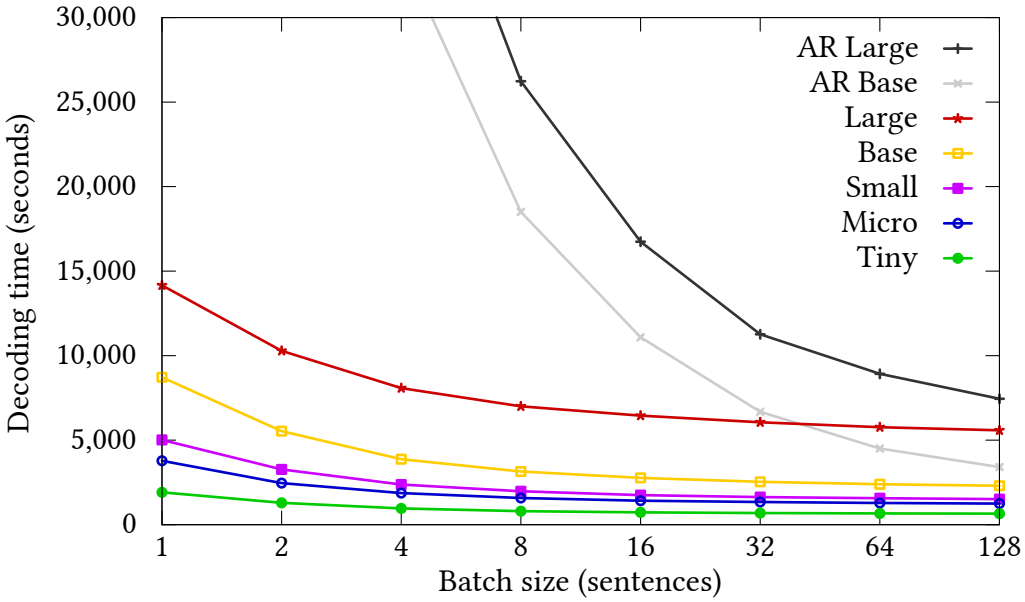
From this point of view, the optimal scenario for NAR models is a situation in which the system runs in online mode, i.e. with a batch size of 1, or with a batch size of a few sentences. On a faster GPU with more threads, AR models need a larger batch size to meet the speed of NAR models.

We measured the GPU decoding time both with and without shortlisting. However, the differences were minimal. Therefore, we only report decoding times without using lexical shortlists.

**CPU Latency and Throughput.** In Figure 5.5, we show the CPU decoding times of the trained models using 36 CPU cores with different batching settings. We see similar trends to GPU decoding – the NAR models are faster with smaller batch sizes. The AR models eventually match the decoding speed of the NAR models as the batch size increases. However, there is a considerable difference between the large and base models in both AR and NAR variants.

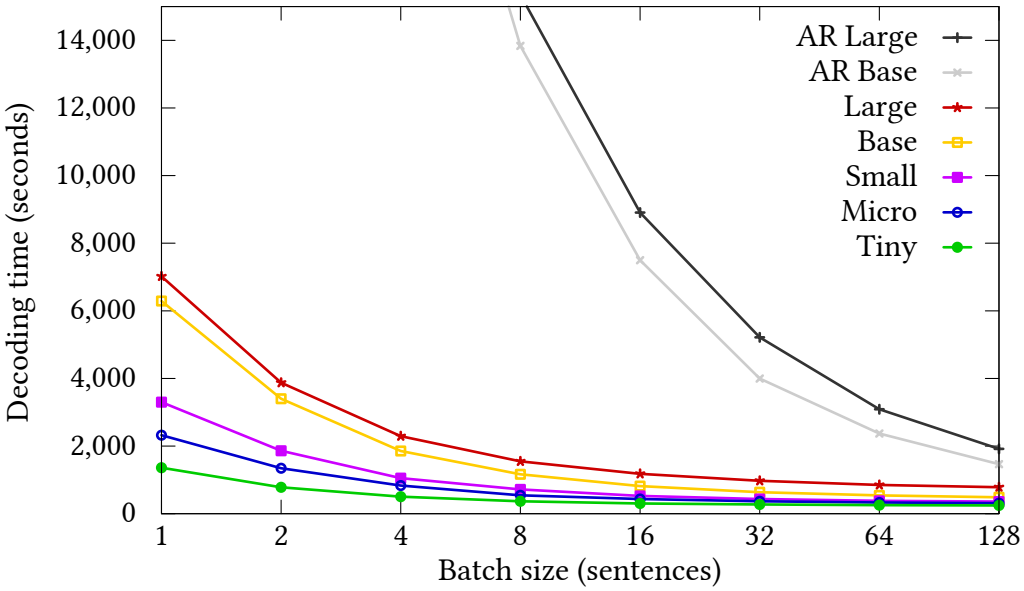
We also notice that increasing the batch size can slow the decoding speed down. There may be several reasons for this behavior. First, the size of the shortlist grows proportionally with the batch size, as there are more possible target words. Second, when the batch is too large, much of the computation is wasted on the padded positions in shorter sentences (as explained in Section 2.3.1). These issues are not evident in GPU decoding due to a higher level of parallelism.

It is apparent from the data table in Figure 5.5 that the use of a lexical shortlist improves the decoding speed of both AR and NAR models. For clarity, we only plot the measured decoding times with shortlisting. We see that changing the batch size has a similar effect in both cases, perhaps with the exception of the micro and tiny models, which benefit greatly from the shortlist in combination with a small batch size.




| Batch size | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|---|---|---|
| AR – Large | | | 44,838 | 26,231 | 16,741 | 11,266 | 8,926 | 7,449 |
| AR – Base | | 61,965 | 33,957 | 18,503 | 11,087 | 6,677 | 4,503 | 3,409 |
| Large | 14,157 | 10,285 | 8,079 | 7,003 | 6,452 | 6,056 | 5,767 | 5,581 |
| Base | 8,720 | 5,532 | 3,875 | 3,146 | 2,769 | 2,535 | 2,394 | 2,303 |
| Small | 5,017 | 3,270 | 2,376 | 1,977 | 1,748 | 1,635 | 1,564 | 1,512 |
| Micro | 3,781 | 2,462 | 1,871 | 1,579 | 1,418 | 1,343 | 1,282 | 1,248 |
| Tiny | 1,916 | 1,295 | 960 | 803 | 730 | 687 | 666 | 652 |

Figure 5.3: Wall times to translate one million sentences (in seconds) on a single Nvidia *Pascal* P100 GPU with different batch size settings.




| Batch size | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|---|---|---|
| AR – Large | | 53,902 | 29,369 | 15,351 | 8,907 | 5,216 | 3,090 | 1,918 |
| AR – Base | | 47,145 | 25,745 | 13,836 | 7,498 | 3,997 | 2,371 | 1,465 |
| Large | 7,020 | 3,874 | 2,292 | 1,547 | 1,179 | 973 | 850 | 782 |
| Base | 6,289 | 3,400 | 1,854 | 1,166 | 816 | 635 | 542 | 485 |
| Small | 3,300 | 1,860 | 1,051 | 717 | 526 | 434 | 380 | 357 |
| Micro | 2,322 | 1,345 | 833 | 544 | 433 | 367 | 332 | 311 |
| Tiny | 1,360 | 780 | 503 | 367 | 301 | 273 | 252 | 243 |

Figure 5.4: Wall times to translate one million sentences (in seconds) on a single Nvidia *Ampere* A100 GPU with different batch size settings.

| | Translation quality | | | Decoding time (seconds) | | |
|---|---|---|---|---|---|---|
| | ChrF | COMET | BLEU | GPU, b>1 | GPU, b=1 | CPU, b>1 |
| Edinburgh base | 61.5 | 0.527 | 55.3 | 140 | 16,851 | 500 |
| AR – Large (teacher) | 59.2 | 0.411 | 50.5 | 1,918 | *> 24h* | 9,090 |
| AR – Base (student) | 59.5 | 0.455 | 51.6 | 1,465 | *> 24h* | 2,587 |
| NAR – Large | 58.6 | 0.149 | 47.8 | 782 | 7,020 | 7,434 |
| NAR – Micro | 57.3 | -0.008 | 43.5 | 311 | 2,322 | 897 |

Table 5.9: Comparison of our models with the Edinburgh "base" model submitted to the WMT Efficiency Shared Task (Behnke et al., 2021). Columns denoted *b>1* show the best result using batching, *b=1* is measured with a single sentence in the batch. CPU times were measured using 36 CPU cores.

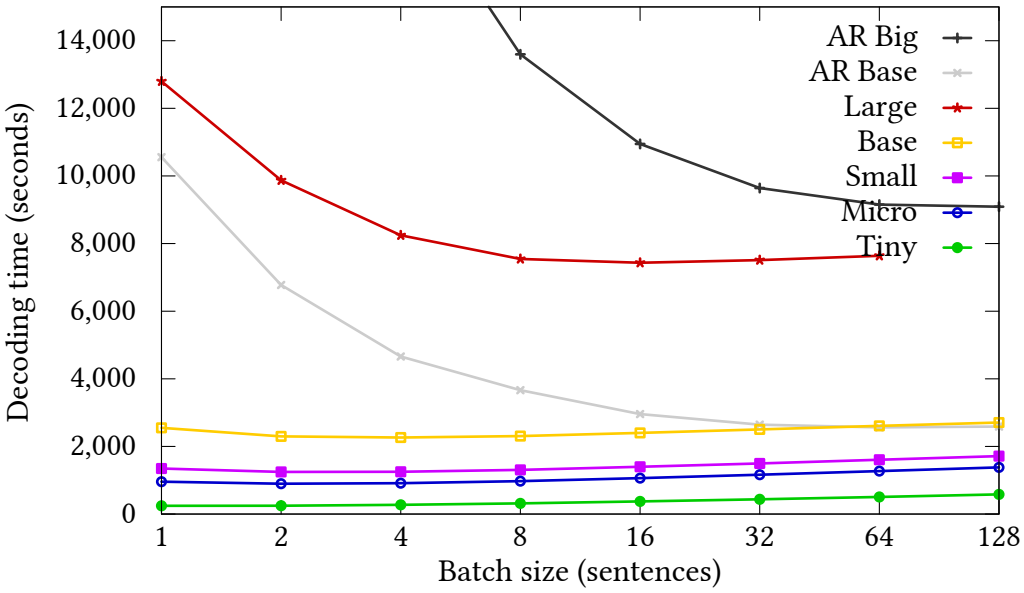
In case of the NAR model with batch size of 128, we ran across an out-of-memory error due to a limitation in the Marian implementation. When using multiple CPUs for the decoding, the program makes a copy of the whole model for each CPU core. When the batch size is too large, this will eventually fill the whole RAM, and the process is killed.

We tried to measure single-core CPU latency, but the decoding speed was too slow to compete with efficiency task submissions. Based on the decoding time on a smaller sample of the data, we estimate the time to translate the whole dataset would take around 100 hours using the large NAR model and around 15 hours using the tiny model.

### 5.3.3 Discussion

Tables 5.7 and 5.8 in Section 5.3.1 show that we achieve state-of-the-art scores in non-autoregressive translation on the WMT 14 test set, outperforming both single-step and iterative methods (see the results overview of the related work in Table 3.1 in Section 3.6). However, the results on the WMT 21 test set in Table 5.6 show that there is still a great deal of room for improvement, especially when looking at the COMET scores. We believe that these findings should motivate future research not to evaluate translation quality exclusively on the WMT 14 test set.

We compare our models with one of the best performing submissions in the efficiency task – the University of Edinburgh's "base" model (Behnke et al., 2021) – in Table 5.9. It is clear from the comparison that the Edinburgh autoregressive model is superior to our NAR models in most regards. One exception is the GPU decoding latency. As we discuss in Section 3.6, these conditions are the only scenario considered in most of the related studies.




| Batch size | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|---|---|---|
| Full output | | | | | | | | |
| AR – Large | 52,087 | 33,189 | 23,478 | 16,731 | 12,166 | 10,151 | 9,370 | 9,244 |
| AR – Base | 16,293 | 9,727 | 7,191 | 5,420 | 3,635 | 2,925 | 2,707 | 2,664 |
| Large | 14,542 | 11,320 | 9,534 | 8,657 | 8,357 | 8,247 | 8,238 | |
| Base | 3,508 | 3,126 | 2,979 | 2,927 | 2,921 | 2,920 | 2,934 | 2,948 |
| Small | 2,346 | 2,079 | 1,975 | 1,933 | 1,921 | 1,921 | 1,936 | 1,950 |
| Micro | 1,952 | 1,728 | 1,641 | 1,600 | 1,588 | 1,587 | 1,607 | 1,627 |
| Tiny | 784 | 719 | 697 | 687 | 685 | 694 | 701 | 719 |
| Shortlist | | | | | | | | |
| AR – Large | 41,168 | 27,977 | 18,914 | 13,597 | 10,946 | 9,643 | 9,154 | 9,090 |
| AR – Base | 10,555 | 6,776 | 4,660 | 3,664 | 2,957 | 2,643 | 2,564 | 2,587 |
| Large | 12,799 | 9,870 | 8,245 | 7,545 | 7,434 | 7,511 | 7,639 | |
| Base | 2,549 | 2,298 | 2,263 | 2,306 | 2,399 | 2,503 | 2,609 | 2,707 |
| Small | 1,346 | 1,246 | 1,250 | 1,306 | 1,399 | 1,497 | 1,606 | 1,714 |
| Micro | 958 | 897 | 913 | 974 | 1,062 | 1,163 | 1,271 | 1,380 |
| Tiny | 244 | 246 | 273 | 316 | 373 | 437 | 506 | 582 |

Figure 5.5: Wall times to translate one million sentences (in seconds) on 36 CPU cores with different batch size settings. The graph shows the decoding times with shortlisting.

# 6
# Conclusions

In this thesis, we explored the possibilities of non-autoregressive (NAR) models for neural machine translation (NMT). We described the commonly used NMT models and showed techniques to effectively train them and to use them for translation.

We presented a comprehensive survey of the related research efforts that have been made so far in this rapidly developing field. We described NAR methods based on latent variables, iterative refinement, or on the relaxed alignment between the output predictions and the ground-truth labels.

We pointed out some of the drawbacks in related NAR approaches, such as using weaker baselines both in terms of translation quality and decoding speed, and identified some of the flaws in the evaluation methodology.

In the first of the two experimental chapters (Chapter 4), we proposed a novel method for NAR NMT based on connectionist temporal classification (CTC) which has been adopted by the NAR research community as one of the basic approaches since its original publication.

In Chapter 5, we aim at improving our CTC-based approach by using knowledge distillation using a strong autoregressive (AR) teacher model. We compare our results to both the WMT 14 benchmark and the results of the WMT 21 Efficiency Shared Task (Heafield et al., 2021). We find that even though our models are among the best performing on WMT 14, there is a large room for improvement when we compare them to highly optimized autoregressive models submitted to the efficiency task.

To conclude, over the past few years, the research community has set its hopes on the NAR models because, in theory, the decoding has lower time complexity. Moreover, the literature on this topic often claims that NAR models already match the performance of their AR counterparts. However, our findings suggest that, in fair comparison with strong efficient baselines, many research studies concluding that

NAR models are superior over AR models may be overclaiming. To avoid these problems in the future, research of NAR models should take into account findings of the efficiency task, evaluate the translation quality also on newer test sets, and measure the decoding speed under various conditions, such as batched GPU and CPU decoding.

## Future Work

Based on our results, future research can be conducted in two main directions – focusing either on the translation quality of the non-autoregressive NMT (NAT) models or on the decoding speed. As we have shown in a case study with lexical shortlists, optimization techniques commonly applied to AR models have the potential to help NAR models as well. These techniques include quantization or pruning of parts of the model, such as the attention heads.

Knowledge distillation has been shown to amplify gender bias already present the teacher model (Vamvas and Sennrich, 2021). This phenomenon is especially relevant for NAT models since knowledge distillation is an integral part of the training. Future work could address the problem of gender bias in these models. A qualitative analysis of other aspects could be also considered, for example, measuring the morphological quality or the performance on different domains or in low-resource scenarios.

# Bibliography


Abadi, M. et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Available at: https://www.tensorflow.org/. Software available from tensorflow.org.

Aharoni, R. – Johnson, M. – Firat, O. Massively Multilingual Neural Machine Translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, p. 3874–3884, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1388. Available at: https://aclanthology.org/N19-1388.

Akhbardeh, F. et al. Findings of the 2021 Conference on Machine Translation (WMT21). In *Proceedings of the Sixth Conference on Machine Translation*, p. 1–93, Online, November 2021. Association for Computational Linguistics. Available at: https://aclanthology.org/2021.wmt-1.1.

Amodei, D. – Ananthanarayanan, S. – Anubhai, R. – Bai, J. – Battenberg, E. – Case, C. – Casper, J. – Catanzaro, B. – Cheng, Q. – Chen, G. – others. Deep Speech 2: End-to-end Speech Recognition in English and Mandarin. In *International conference on machine learning*, p. 173–182. PMLR, 2016.

Ba, L. J. – Kiros, R. – Hinton, G. E. Layer Normalization. *CoRR*. 2016, abs/1607.06450. ISSN 2331-8422.

Bahdanau, D. – Cho, K. – Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR*. 2014, abs/1409.0473. ISSN 2331-8422.

Barrault, L. – Bojar, O. – Costa-jussà, M. R. – Federmann, C. – Fishel, M. – Graham, Y. – Haddow, B. – Huck, M. – Koehn, P. – Malmasi, S. – Monz, C. – Müller, M. – Pal, S. – Post, M. – Zampieri, M. Findings of the 2019 Conference on Machine Translation (WMT19). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, p. 1–61, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5301. Available at: https://aclanthology.org/W19-5301.

Barrault, L. et al. Findings of the 2020 Conference on Machine Translation (WMT20). In *Proceedings of the Fifth Conference on Machine Translation*, p. 1–55, Online, November 2020. Association for Computational Linguistics. Available at: https://aclanthology.org/2020.wmt-1.1.

Behnke, M. – Bogoychev, N. – Aji, A. F. – Heafield, K. – Nail, G. – Zhu, Q. – Tchistiakova, S. – Linde, J. – Chen, P. – Kashyap, S. – Grundkiewicz, R. Efficient Machine Translation with Model Pruning and Quantization. In *Proceedings of the Conference on Machine Translation at the 2021 Conference on Empirical Methods in Natural Language Processing*, Punta Cana, Dominican Republic, November 2021. Available at: https://kheafield.com/papers/edinburgh/wmt21-speed.pdf.

Bengio, S. – Vinyals, O. – Jaitly, N. – Shazeer, N. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 1*, p. 1171–1179, 2015.

Bengio, Y. – Ducharme, R. – Vincent, P. – Jauvin, C. A Neural Probabilistic Language Model. *Journal of machine learning research*. 2003, 3, Feb, p. 1137–1155.

Bogoychev, N. – Grundkiewicz, R. – Aji, A. F. – Behnke, M. – Heafield, K. – Kashyap, S. – Farsarakis, E.-I. – Chudyk, M. Edinburgh's Submissions to the 2020 Machine Translation Efficiency Task. In *Proceedings of the Fourth Workshop on Neural Generation and Translation*, p. 218–224, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.ngt-1.26. Available at: https://aclanthology.org/2020.ngt-1.26.

Bojar, O. – Tamchyna, A. Improving Translation Model by Monolingual Data. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, p. 330–336, Edinburgh, Scotland, July 2011. Association for Computational Linguistics. Available at: https://aclanthology.org/W11-2138.

Bojar, O. – Kos, K. – Mareček, D. Tackling Sparse Data Issue in Machine Translation Evaluation. In *Proceedings of the ACL 2010 Conference Short Papers*, p. 86–91, Uppsala, Sweden, July 2010. Association for Computational Linguistics. Available at: https://aclanthology.org/P10-2016.

Bojar, O. – Buck, C. – Callison-Burch, C. – Federmann, C. – Haddow, B. – Koehn, P. – Monz, C. – Post, M. – Soricut, R. – Specia, L. Findings of the 2013 Workshop on Statistical Machine Translation. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, p. 1–44, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. Available at: https://aclanthology.org/W13-2201.

Bojar, O. – Buck, C. – Federmann, C. – Haddow, B. – Koehn, P. – Leveling, J. – Monz, C. – Pecina, P. – Post, M. – Saint-Amand, H. – Soricut, R. – Specia, L. – Tamchyna, A. Findings of the 2014 Workshop on Statistical Machine Translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, p. 12–58, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-3302. Available at: https://aclanthology.org/W14-3302.

Bojar, O. – Chatterjee, R. – Federmann, C. – Haddow, B. – Huck, M. – Hokamp, C. – Koehn, P. – Logacheva, V. – Monz, C. – Negri, M. – Post, M. – Scarton, C. – Specia, L. – Turchi, M. Findings of the 2015 Workshop on Statistical Machine Translation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, p. 1–46, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-3001. Available at: https://aclanthology.org/W15-3001.

Bojar, O. et al. Findings of the 2016 Conference on Machine Translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, p. 131–198, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/W16-2301. Available at: https://aclanthology.org/W16-2301.

Bojar, O. – Federmann, C. – Fishel, M. – Graham, Y. – Haddow, B. – Koehn, P. – Monz, C. Findings of the 2018 Conference on Machine Translation (WMT18). In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, p. 272–303, Belgium, Brussels, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-6401. Available at: https://aclanthology.org/W18-6401.

Brown, P. F. – Cocke, J. – Della Pietra, S. A. – Della Pietra, V. J. – Jelinek, F. – Lafferty, J. D. – Mercer, R. L. – Roossin, P. S. A Statistical Approach to Machine Translation. *Computational Linguistics*. 1990, 16, 2, p. 79–85. Available at: https://aclanthology.org/J90-2002.

Brown, P. F. – Della Pietra, S. A. – Della Pietra, V. J. – Mercer, R. L. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*. 1993, 19, 2, p. 263–311. Available at: https://aclanthology.org/J93-2003.

Callison-Burch, C. – Osborne, M. – Koehn, P. Re-evaluating the Role of Bleu in Machine Translation Research. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, p. 249–256, Trento, Italy, April 2006. Association for Computational Linguistics. Available at: https://aclanthology.org/E06-1032.

Caswell, I. – Chelba, C. – Grangier, D. Tagged Back-Translation. In *Proceedings of the Fourth Conference on Machine Translation (Volume 1: Research Papers)*, p. 53–63, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5206. Available at: https://aclanthology.org/W19-5206.

Chen, P. – Helcl, J. – Germann, U. – Burchell, L. – Bogoychev, N. – Barone, A. V. M. – Waldendorf, J. – Birch, A. – Heafield, K. The University of Edinburgh's English-German and English-Hausa Submissions to the WMT21 News Translation Task. In *Proceedings of the Sixth Conference on Machine Translation*. Association for Computational Linguistics, 2021.

Cho, K. – Merriënboer, B. – Bahdanau, D. – Bengio, Y. 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*, p. 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-4012. Available at: https://aclanthology.org/W14-4012.

Chung, J. – Cho, K. – Bengio, Y. A Character-level Decoder without Explicit Segmentation for Neural Machine Translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, p. 1693–1703, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1160. Available at: https://aclanthology.org/P16-1160.

Collobert, R. – Weston, J. Fast Semantic Extraction Using a Novel Neural Network Architecture. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, p. 560–567, Prague, Czech Republic, June 2007. Association for Computational Linguistics. Available at: https://aclanthology.org/P07-1071.

Conneau, A. – Lample, G. Cross-Lingual Language Model Pretraining. *Advances in Neural Information Processing Systems*. 2019, 32, p. 7059–7069.

Daumé, H. – Langford, J. – Marcu, D. Search-based structured prediction. *Machine learning*. 2009, 75, 3, p. 297–325.

Devlin, J. – Chang, M.-W. – Lee, K. – Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, p. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. Available at: https://aclanthology.org/N19-1423.

Dostert, L. E. The Georgetown-IBM Experiment. *1955). Machine translation of languages. John Wiley & Sons, New York*. 1955, p. 124–135.

Du, C. – Tu, Z. – Jiang, J. Order-Agnostic Cross Entropy for Non-Autoregressive Machine Translation, 2021.

Duchi, J. – Hazan, E. – Singer, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of machine learning research*. 2011, 12, 7.

Dupont, Q. The Cryptological Origins of Machine Translation, from al-Kindi to Weaver. *amodern*. 01 2017.

Dyer, C. – Chahuneau, V. – Smith, N. A. A Simple, Fast, and Effective Reparameterization of IBM Model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, p. 644–648, Atlanta, Georgia, June 2013. Association for Computational Linguistics. Available at: https://aclanthology.org/N13-1073.

Elman, J. L. Finding Structure in Time. *Cognitive Science*. 1990, 14, 2, p. 179–211. ISSN 1551-6709.

Esplà, M. – Forcada, M. – Ramírez-Sánchez, G. – Hoang, H. ParaCrawl: Web-scale parallel corpora for the languages of the EU. In *Proceedings of Machine Translation Summit XVII: Translator, Project and User Tracks*, p. 118–119, Dublin, Ireland, August 2019. European Association for Machine Translation. Available at: https://aclanthology.org/W19-6721.

Eyben, F. – Wöllmer, M. – Schuller, B. – Graves, A. From Speech to Letters – Using a Novel Neural Network Architecture for Grapheme Based ASR. In *2009 IEEE Workshop on Automatic Speech Recognition & Understanding*, p. 376–380. IEEE, 2009.

Gao, Y. – Nikolov, N. I. – Hu, Y. – Hahnloser, R. H. Character-Level Translation with Self-attention. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, p. 1591–1604, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.145. Available at: https://aclanthology.org/2020.acl-main.145.

Gehring, J. – Auli, M. – Grangier, D. – Yarats, D. – Dauphin, Y. N. Convolutional Sequence to Sequence Learning. In *International Conference on Machine Learning*, p. 1243–1252, Sydney, Australia, August 2017. PMLR.

Gers, F. A. – Schmidhuber, J. – Cummins, F. Learning to forget: Continual prediction with LSTM. *Neural computation*. 2000, 12, 10, p. 2451–2471.

Ghazvininejad, M. – Levy, O. – Liu, Y. – Zettlemoyer, L. Mask-Predict: Parallel Decoding of Conditional Masked Language Models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, p. 6112–6121, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1633. Available at: https://aclanthology.org/D19-1633.

Ghazvininejad, M. – Karpukhin, V. – Zettlemoyer, L. – Levy, O. Aligned Cross Entropy for Non-Autoregressive Machine Translation. In III, H. D. – Singh, A. (Ed.) *Proceedings of the 37th International Conference on Machine Learning*, 119 / *Proceedings of Machine Learning Research*, p. 3515–3523. PMLR, 13–18 Jul 2020a. Available at: http://proceedings.mlr.press/v119/ghazvininejad20a.html.

Ghazvininejad, M. – Levy, O. – Zettlemoyer, L. Semi-Autoregressive Training Improves Mask-Predict Decoding. *ArXiv*. 2020b, abs/2001.08785.

Glorot, X. – Bengio, Y. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In Teh, Y. W. – Titterington, M. (Ed.) *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 9 / *Proceedings of Machine Learning Research*, p. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. Available at: https://proceedings.mlr.press/v9/glorot10a.html.

Goodfellow, I. – Bengio, Y. – Courville, A. *Deep learning*. MIT press, 2016.

Graves, A. – Jaitly, N. Towards End-to-End Speech Recognition with Recurrent Neural Networks. In *International conference on machine learning*, p. 1764–1772. PMLR, 2014.

Graves, A. – Fernández, S. – Gomez, F. – Schmidhuber, J. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *Proceedings of the 23rd International Conference on Machine Learning*, p. 369–376, Pittsburgh, PA, USA, June 2006. JMLR.org.

Gu, J. – Kong, X. Fully Non-autoregressive Neural Machine Translation: Tricks of the Trade. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, p. 120–133, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.11. Available at: https://aclanthology.org/2021.findings-acl.11.

Gu, J. – Bradbury, J. – Xiong, C. – Li, V. O. K. – Socher, R. Non-Autoregressive Neural Machine Translation. In *6th International Conference on Learning Representations, ICLR 2018*, Vancouver, BC, Canada, April 2018. Available at: https://openreview.net/forum?id=B1l8BtlCb.

Gu, J. – Wang, C. – Zhao, J. Levenshtein Transformer. In Wallach, H. – Larochelle, H. – Beygelzimer, A. – Alché-Buc, F. – Fox, E. – Garnett, R. (Ed.) *Advances in Neural Information Processing Systems*, 32, p. 11181–11191. Curran Associates, Inc., 2019. Available at: https://proceedings.neurips.cc/paper/2019/file/675f9820626f5bc0afb47b57890b466e-Paper.pdf.

Guo, J. – Xu, L. – Chen, E. Jointly Masked Sequence-to-Sequence Model for Non-Autoregressive Neural Machine Translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, p. 376–385, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.36. Available at: https://aclanthology.org/2020.acl-main.36.

Haddow, B. – Bawden, R. – Barone, A. V. M. – Helcl, J. – Birch, A. Survey of Low-Resource Machine Translation. *arXiv preprint arXiv:2109.00486*. 2021.

Heafield, K. KenLM: Faster and Smaller Language Model Queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, p. 187–197, Edinburgh, Scotland, July 2011. Association for Computational Linguistics. Available at: https://aclanthology.org/W11-2123.

Heafield, K. – Hayashi, H. – Oda, Y. – Konstas, I. – Finch, A. – Neubig, G. – Li, X. – Birch, A. Findings of the Fourth Workshop on Neural Generation and Translation. In *Proceedings of the Fourth Workshop on Neural Generation and Translation*, p. 1–9, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.ngt-1.1. Available at: https://aclanthology.org/2020.ngt-1.1.

Heafield, K. – Zhu, Q. – Grundkiewicz, R. Findings of the WMT 2021 Shared Task on Efficient Translation. In *Proceedings of the Conference on Machine Translation at the 2021 Conference on Empirical Methods in Natural Language Processing*, Punta Cana, Dominican Republic, November 2021. Available at: https://kheafield.com/papers/edinburgh/wmt21-speedtask.pdf.

Helcl, J. – Libovický, J. – Kocmi, T. – Musil, T. – Cífka, O. – Variš, D. – Bojar, O. Neural Monkey: The Current State and Beyond. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*, p. 168–176, Boston, MA, March 2018. Association for Machine Translation in the Americas. Available at: https://aclanthology.org/W18-1816.

Helcl, J. – Libovický, J. Neural Monkey: An Open-source Tool for Sequence Learning. *The Prague Bulletin of Mathematical Linguistics*. Apr 2017, 107, 1, p. 5–17. ISSN 0032-6585.

Hochreiter, S. – Schmidhuber, J. Long Short-Term Memory. *Neural Computation*. 1997, 9, 8, p. 1735–1780. ISSN 0899-7667.

Huang, C. – Zhou, H. – Zaïane, O. R. – Mou, L. – Li, L. Non-Autoregressive Translation with Layer-Wise Prediction and Deep Supervision, 2021.

Huang, L. – Fayong, S. – Guo, Y. 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*, p. 142–151, Montréal, Canada, June 2012. Association for Computational Linguistics. Available at: https://aclanthology.org/N12-1015.

Jean, S. – Cho, K. – Memisevic, R. – Bengio, Y. On Using Very Large Target Vocabulary for Neural Machine Translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, p. 1–10, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1001. Available at: https://aclanthology.org/P15-1001.

Joulin, A. – Grave, E. – Bojanowski, P. – Douze, M. – Jégou, H. – Mikolov, T. FastText.zip: Compressing Text Classification Models. *arXiv preprint arXiv:1612.03651*. 2016.

Joulin, A. – Grave, E. – Bojanowski, P. – Mikolov, T. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, p. 427–431, Valencia, Spain, April 2017. Association for Computational Linguistics. Available at: https://aclanthology.org/E17-2068.

Junczys-Dowmunt, M. Dual Conditional Cross-Entropy Filtering of Noisy Parallel Corpora. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, p. 888–895, Belgium, Brussels, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-6478. Available at: https://aclanthology.org/W18-6478.

Junczys-Dowmunt, M. – Grundkiewicz, R. – Dwojak, T. – Hoang, H. – Heafield, K. – Neckermann, T. – Seide, F. – Germann, U. – Aji, A. F. – Bogoychev, N. – Martins, A. F. T. – Birch, A. Marian: Fast Neural Machine Translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, p. 116–121, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-4020. Available at: https://aclanthology.org/P18-4020.

Kaiser, L. – Bengio, S. – Roy, A. – Vaswani, A. – Parmar, N. – Uszkoreit, J. – Shazeer, N. Fast Decoding in Sequence Models Using Discrete Latent Variables. In Dy, J. – Krause, A. (Ed.) *Proceedings of the 35th International Conference on Machine Learning*, 80 / *Proceedings of Machine Learning Research*, p. 2390–2399, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. Available at: http://proceedings.mlr.press/v80/kaiser18a.html.

Kasai, J. – Cross, J. – Ghazvininejad, M. – Gu, J. Non-Autoregressive Machine Translation with Disentangled Context Transformer. In III, H. D. – Singh, A. (Ed.) *Proceedings of the 37th International Conference on Machine Learning*, 119 / *Proceedings of Machine Learning Research*, p. 5144–5155. PMLR, 13–18 Jul 2020. Available at: http://proceedings.mlr.press/v119/kasai20a.html.

Kasner, Z. Incorporating Language Models into Non-autoregressive Neural Machine Translation. Master's thesis, Czech Technical University, 2020.

Kasner, Z. – Libovický, J. – Helcl, J. Improving Fluency of Non-Autoregressive Machine Translation, 2020.

Kim, Y. – Rush, A. M. Sequence-Level Knowledge Distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, p. 1317–1327, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1139. Available at: https://aclanthology.org/D16-1139.

Kim, Y. J. – Junczys-Dowmunt, M. – Hassan, H. – Fikri Aji, A. – Heafield, K. – Grundkiewicz, R. – Bogoychev, N. From Research to Production and Back: Ludicrously Fast Neural Machine Translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, p. 280–288, Hong Kong, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5632. Available at: https://aclanthology.org/D19-5632.

Kingma, D. P. – Ba, J. Adam: A Method for Stochastic Optimization. *CoRR*. 2014, abs/1412.6980. ISSN 2331-8422.

Kocmi, T. – Bojar, O. Curriculum Learning and Minibatch Bucketing in Neural Machine Translation. In *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2017*, p. 379–386, Varna, Bulgaria, September 2017. INCOMA Ltd. doi: 10.26615/978-954-452-049-6_050. Available at: https://doi.org/10.26615/978-954-452-049-6_050.

Kocmi, T. – Federmann, C. – Grundkiewicz, R. – Junczys-Dowmunt, M. – Matsushita, H. – Menezes, A. To ship or not to ship: An extensive evaluation of automatic metrics for machine translation. *arXiv preprint arXiv:2107.10821*. 2021.

Koehn, P. Europarl: A Parallel Corpus for Statistical Machine Translation. In *Proceedings of Machine Translation Summit X: Papers*, p. 79–86, Phuket, Thailand, September 13-15 2005. Available at: https://aclanthology.org/2005.mtsummit-papers.11.

Koehn, P. *Statistical Machine Translation*. Cambridge University Press, 2009. Available at: http://www.statmt.org/book/. ISBN 978-0521874151.

Koehn, P. – Och, F. J. – Marcu, D. Statistical Phrase-Based Translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, p. 127–133, 2003. Available at: https://aclanthology.org/N03-1017.

Koehn, P. – Hoang, H. – Birch, A. – Callison-Burch, C. – Federico, M. – Bertoldi, N. – Cowan, B. – Shen, W. – Moran, C. – Zens, R. – Dyer, C. – Bojar, O. – Constantin, A. – Herbst, E. Moses: Open Source Toolkit for Statistical Machine Translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, p. 177–180, Prague, Czech Republic, June 2007. Association for Computational Linguistics. Available at: https://aclanthology.org/P07-2045.

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)*, p. 66–75, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1007. Available at: https://aclanthology.org/P18-1007.

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*, p. 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. Available at: https://aclanthology.org/D18-2012.

Kuhn, H. W. The Hungarian method for the assignment problem. *Naval research logistics quarterly*. 1955, 2, 1-2, p. 83–97.

Lee, J. – Cho, K. – Hofmann, T. Fully Character-Level Neural Machine Translation without Explicit Segmentation. *Transactions of the Association for Computational Linguistics*. 2017, 5, p. 365–378. doi: 10.1162/tacl_a_00067. Available at: https://aclanthology.org/Q17-1026.

Lee, J. – Mansimov, E. – Cho, K. Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, p. 1173–1182, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1149. Available at: https://aclanthology.org/D18-1149.

Li, Z. – Lin, Z. – He, D. – Tian, F. – Qin, T. – Wang, L. – Liu, T.-Y. Hint-Based Training for Non-Autoregressive Machine Translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, p. 5708–5713, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1573. Available at: https://aclanthology.org/D19-1573.

Libovický, J. – Helcl, J. Attention Strategies for Multi-Source Sequence-to-Sequence Learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, p. 196–202, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-2031. Available at: https://aclanthology.org/P17-2031.

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*, p. 3016–3021, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1336. Available at: https://aclanthology.org/D18-1336.

Libovický, J. – Helcl, J. – Mareček, D. Input Combination Strategies for Multi-Source Transformer Decoder. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, p. 253–260, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-6326. Available at: https://aclanthology.org/W18-6326.

Liu, C. – Zhang, Q. – Zhang, X. – Singh, K. – Saraf, Y. – Zweig, G. Multilingual Graphemic Hybrid ASR with Massive Data Augmentation. In *Proceedings of the 1st Joint Workshop on Spoken Language Technologies for Under-resourced languages (SLTU) and Collaboration and Computing for Under-Resourced Languages (CCURL)*, p. 46–52, Marseille, France, May 2020. European Language Resources association. Available at: https://aclanthology.org/2020.sltu-1.7. ISBN 979-10-95546-35-1.

Liwicki, M. – Graves, A. – Fernàndez, S. – Bunke, H. – Schmidhuber, J. A Novel Approach to On-Line Handwriting Recognition Based on Bidirectional Long Short-Term Memory Networks. In *Proceedings of the 9th International Conference on Document Analysis and Recognition, ICDAR 2007*, 2007.

Luong, T. – Pham, H. – Manning, C. D. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, p. 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1166. Available at: https://aclanthology.org/D15-1166.

Ma, X. – Zhou, C. – Li, X. – Neubig, G. – Hovy, E. FlowSeq: Non-Autoregressive Conditional Sequence Generation with Generative Flow. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, p. 4282–4292, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1437. Available at: https://aclanthology.org/D19-1437.

Mansimov, E. – Wang, A. – Cho, K. A Generalized Framework of Sequence Generation with Application to Undirected Sequence Models. *CoRR*. 2019, abs/1905.12790. Available at: http://arxiv.org/abs/1905.12790.

Marie, B. – Rubino, R. – Fujita, A. Tagged Back-translation Revisited: Why Does It Really Work? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, p. 5990–5997, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.532. Available at: https://aclanthology.org/2020.acl-main.532.

Mathur, N. – Baldwin, T. – Cohn, T. Tangled up in BLEU: Reevaluating the Evaluation of Automatic Machine Translation Evaluation Metrics. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, p. 4984–4997, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.448. Available at: https://aclanthology.org/2020.acl-main.448.

Miceli Barone, A. V. – Helcl, J. – Sennrich, R. – Haddow, B. – Birch, A. Deep architectures for Neural Machine Translation. In *Proceedings of the Second Conference on Machine Translation*, p. 99–107, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4710. Available at: https://aclanthology.org/W17-4710.

Mikolov, T. – Sutskever, I. – Chen, K. – Corrado, G. S. – Dean, J. Distributed Representations of Words and Phrases and Their Compositionality. In *Advances in neural information processing systems*, p. 3111–3119, 2013.

Nagao, M. A framework of a mechanical translation between Japanese and English by analogy principle. *Artificial and human intelligence*. 1984, p. 351–354.

Papineni, K. – Roukos, S. – Ward, T. – Zhu, W.-J. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, p. 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. Available at: https://aclanthology.org/P02-1040.

Popel, M. – Bojar, O. Training Tips for the Transformer Model. *The Prague Bulletin of Mathematical Linguistics*. April 2018, 110, p. 43–70. ISSN 0032-6585. doi: 10.2478/pralin-2018-0002. Available at: https://ufal.mff.cuni.cz/pbml/110/art-popel-bojar.pdf.

Popel, M. – Tomkova, M. – Tomek, J. – Kaiser, Ł. – Uszkoreit, J. – Bojar, O. – Žabokrtský, Z. Transforming Machine Translation: A Deep Learning System Reaches News Translation Quality Comparable to Human Professionals. *Nature Communications*. 2020, 11, 4381, p. 1–15. ISSN 2041-1723. doi: 10.1038/s41467-020-18073-9. Available at: https://www.nature.com/articles/s41467-020-18073-9.

Popović, M. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, p. 392–395, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-3049. Available at: https://aclanthology.org/W15-3049.

Post, M. A Call for Clarity in Reporting BLEU Scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, p. 186–191, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-6319. Available at: https://aclanthology.org/W18-6319.

Qian, L. – Zhou, H. – Bao, Y. – Wang, M. – Qiu, L. – Zhang, W. – Yu, Y. – Li, L. Glancing Transformer for Non-Autoregressive Neural Machine Translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, p. 1993–2003, Online, August 2021a. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.155. Available at: https://aclanthology.org/2021.acl-long.155.

Qian, L. – Zhou, Y. – Zheng, Z. – Zhu, Y. – Lin, Z. – Feng, J. – Cheng, S. – Li, L. – Wang, M. – Zhou, H. The Volctrans GLAT System: Non-autoregressive Translation Meets WMT21, 2021b.

Ran, Q. – Lin, Y. – Li, P. – Zhou, J. Guiding Non-Autoregressive Neural Machine Translation Decoding with Reordering Information. *Proceedings of the AAAI Conference on Artificial Intelligence*. May 2021, 35, 15, p. 13727–13735. Available at: https://ojs.aaai.org/index.php/AAAI/article/view/17618.

Ranzato, M. – Chopra, S. – Auli, M. – Zaremba, W. Sequence Level Training with Recurrent Neural Networks. In Bengio, Y. – LeCun, Y. (Ed.) *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. Available at: http://arxiv.org/abs/1511.06732.

Rei, R. – Stewart, C. – Farinha, A. C. – Lavie, A. COMET: A Neural Framework for MT Evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, p. 2685–2702, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.213. Available at: https://aclanthology.org/2020.emnlp-main.213.

Reiter, E. A Structured Review of the Validity of BLEU. *Computational Linguistics*. 2018, 44, 3, p. 393–401. doi: 10.1162/coli_a_00322. Available at: https://doi.org/10.1162/coli_a_00322.

Rezende, D. – Mohamed, S. Variational Inference with Normalizing Flows. In *International conference on machine learning*, p. 1530–1538. PMLR, 2015.

Rozis, R. – Skadiņš, R. Tilde MODEL - Multilingual Open Data for EU Languages. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, p. 263–265, Gothenburg, Sweden, May 2017. Association for Computational Linguistics. Available at: https://aclanthology.org/W17-0235.

Saharia, C. – Chan, W. – Saxena, S. – Norouzi, M. Non-Autoregressive Machine Translation with Latent Alignments. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, p. 1098–1108, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.83. Available at: https://aclanthology.org/2020.emnlp-main.83.

Schuster, M. – Nakajima, K. Japanese and Korean Voice Search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, p. 5149–5152, 2012. doi: 10.1109/ICASSP.2012.6289079.

Schwenk, H. – Chaudhary, V. – Sun, S. – Gong, H. – Guzmán, F. Wikimatrix: Mining 135M Parallel Sentences in 1620 Language Pairs from Wikipedia. *arXiv preprint arXiv:1907.05791*. 2019.

Sennrich, R. – Haddow, B. – Birch, A. Improving Neural Machine Translation Models with Monolingual Data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, p. 86–96, Berlin, Germany, August 2016a. Association for Computational Linguistics. doi: 10.18653/v1/P16-1009. Available at: https://aclanthology.org/P16-1009.

Sennrich, R. – Haddow, B. – Birch, A. 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)*, p. 1715–1725, Berlin, Germany, August 2016b. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. Available at: https://aclanthology.org/P16-1162.

Shannon, C. E. A mathematical theory of communication. *The Bell system technical journal*. 1948, 27, 3, p. 379–423.

Shao, C. – Zhang, J. – Feng, Y. – Meng, F. – Zhou, J. Minimizing the Bag-of-Ngrams Difference for Non-Autoregressive Neural Machine Translation. *Proceedings of the AAAI Conference on Artificial Intelligence*. Apr. 2020, 34, 01, p. 198–205. doi: 10.1609/aaai.v34i01.5351. Available at: https://ojs.aaai.org/index.php/AAAI/article/view/5351.

Slocum, J. A Survey of Machine Translation: Its History, Current Status and Future Prospects. *Computational Linguistics*. 1985, 11, 1, p. 1–17. Available at: https://aclanthology.org/J85-1001.

Srivastava, N. – Hinton, G. E. – Krizhevsky, A. – Sutskever, I. – Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research*. 2014, 15, 1, p. 1929–1958. ISSN 1532-4435.

Stern, M. – Shazeer, N. – Uszkoreit, J. Blockwise Parallel Decoding for Deep Autoregressive Models. In Bengio, S. – Wallach, H. – Larochelle, H. – Grauman, K. – Cesa-Bianchi, N. – Garnett, R. (Ed.) *Advances in Neural Information Processing Systems*, 31, p. 10086–10095. Curran Associates, Inc., 2018. Available at: https://proceedings.neurips.cc/paper/2018/file/c4127b9194fe8562c64dc0f5bf2c93bc-Paper.pdf.

Stern, M. – Chan, W. – Kiros, J. – Uszkoreit, J. Insertion Transformer: Flexible Sequence Generation via Insertion Operations. In Chaudhuri, K. – Salakhutdinov, R. (Ed.) *Proceedings of the 36th International Conference on Machine Learning*, 97 / *Proceedings of Machine Learning Research*, p. 5976–5985. PMLR, 09–15 Jun 2019. Available at: http://proceedings.mlr.press/v97/stern19a.html.

Sun, Z. – Yang, Y. An EM Approach to Non-Autoregressive Conditional Sequence Generation. In III, H. D. – Singh, A. (Ed.) *Proceedings of the 37th International Conference on Machine Learning*, 119 / *Proceedings of Machine Learning Research*, p. 9249–9258. PMLR, 13–18 Jul 2020. Available at: http://proceedings.mlr.press/v119/sun20c.html.

Sun, Z. – Li, Z. – Wang, H. – He, D. – Lin, Z. – Deng, Z. Fast Structured Decoding for Sequence Models. In Wallach, H. – Larochelle, H. – Beygelzimer, A. – Alché-Buc, F. – Fox, E. – Garnett, R. (Ed.) *Advances in Neural Information Processing Systems*, 32, p. 3016–3026. Curran Associates, Inc., 2019. Available at: https://proceedings.neurips.cc/paper/2019/file/74563ba21a90da13dacf2a73e3ddefa7-Paper.pdf.

Sutskever, I. – Vinyals, O. – Le, Q. V. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, p. 3104–3112, Montreal, Canada, December 2014. Curran Associates, Inc.

Tiedemann, J. Parallel Data, Tools and Interfaces in OPUS. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, p. 2214–2218, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA). Available at: http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf.

Tieleman, T. – Hinton, G. Lecture 6.5—RmsProp: Divide the Gradient by a Running Average of its Recent Magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Turing, A. M. Computing Machinery and Intelligence. *Mind*. 10 1950, LIX, 236, p. 433–460. ISSN 0026-4423. doi: 10.1093/mind/LIX.236.433. Available at: https://doi.org/10.1093/mind/LIX.236.433.

Vamvas, J. – Sennrich, R. Contrastive Conditioning for Assessing Disambiguation in MT: A Case Study of Distilled Bias. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, p. 10246–10265, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. Available at: https://aclanthology.org/2021.emnlp-main.803.

Vaswani, A. – Shazeer, N. – Parmar, N. – Uszkoreit, J. – Jones, L. – Gomez, A. N. – Kaiser, Ł. – Polosukhin, I. Attention is All You Need. In *Advances in Neural Information Processing Systems 30*, p. 6000–6010, Long Beach, CA, USA, December 2017. Curran Associates, Inc.

Voita, E. – Talbot, D. – Moiseev, F. – Sennrich, R. – Titov, I. Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, p. 5797–5808, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1580. Available at: https://aclanthology.org/P19-1580.

Wang, F. – Chen, W. – Yang, Z. – Dong, Q. – Xu, S. – Xu, B. Semi-Supervised Disfluency Detection. In *Proceedings of the 27th International Conference on Computational Linguistics*, p. 3529–3538, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics. Available at: https://aclanthology.org/C18-1299.

Wang, Y. – Tian, F. – He, D. – Qin, T. – Zhai, C. – Liu, T.-Y. Non-Autoregressive Machine Translation with Auxiliary Regularization. *Proceedings of the AAAI Conference on Artificial Intelligence*. Jul. 2019, 33, 01, p. 5377–5384. doi: 10.1609/aaai.v33i01.33015377. Available at: https://ojs.aaai.org/index.php/AAAI/article/view/4476.

Williams, R. J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine learning*. 1992, 8, 3-4, p. 229–256.

Wiseman, S. – Rush, A. M. Sequence-to-Sequence Learning as Beam-Search Optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, p. 1296–1306, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1137. Available at: https://aclanthology.org/D16-1137.

Wu, Y. et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR*. 2016, abs/1609.08144. ISSN 2331-8422.

Zeiler, M. D. Adadelta: An Adaptive Learning Rate Method. *arXiv preprint arXiv:1212.5701*. 2012.

Zhou, C. – Gu, J. – Neubig, G. Understanding Knowledge Distillation in Non-autoregressive Machine Translation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. Available at: https://openreview.net/forum?id=BygFVAEKDH.

Zipf, G. K. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.

Zoph, B. – Yuret, D. – May, J. – Knight, K. Transfer Learning for Low-Resource Neural Machine Translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, p. 1568–1575, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1163. Available at: https://aclanthology.org/D16-1163.

# List of Abbreviations

**AI** artificial intelligence. 2
**AR** autoregressive. 2, 3, 30–34, 36, 37, 39, 40, 43–48, 50, 51, 54, 55, 57–68, 74–76, 78
**ASR** automatic speech recognition. 52
**AXE** aligned cross-entropy. 37, 38, 48, 53

**BLEU** bilingual evaluation understudy. 6, 23, 26, 27, 46, 48, 49, 56, 57, 63, 67, 69, 70, 72, 73, 75–77
**BoN** bag-of-ngrams. 40, 48
**BPE** byte-pair encoding. 8

**ChrF** character F-score. 27, 75, 76
**CMLM** conditional masked language model. 37, 40–44, 48
**CNN** convolutional neural network. 6
**COMET** Crosslingual Optimized Metric for Evaluation of Translation. 27, 75, 76
**CRF** conditional random fields. 45
**CTC** connectionist temporal classification. 3, 36–38, 45, 48, 51–54, 56, 58, 59, 61–64, 66, 67, 72

**DCRF** dynamic-transition conditional random fields. 45, 48
**DisCo** disentangled context. 43, 48

**EM+ODD** expectation-maximization training + optimal deduplicated decoding. 48

**GLAT** Glancing Transformer. 40, 48
**GRU** Gated Recurrent Unit. 10–12

**Hint-NAT** hint-based training for non-autoregressive neural machine translation. 39, 48

**JM-NAT** jointly masked model for non-autoregressive neural machine translation. 43, 48

**LevT** Levenshtein Transformer. 44
**LM** language model. 3, 41, 45, 61–65
**LPD** length parallel decoding. 34
**LSTM** Long Short-Term Memory. 10, 11

**LT** Latent Transformer. 44, 48

**MLM** masked language model. 38, 41
**MT** machine translation. 1, 2, 5, 6, 9, 23, 24, 26, 27, 29, 41, 46, 61

**NAR** non-autoregressive. 2, 3, 30–34, 36, 39, 43–48, 50, 51, 54, 57–61, 63–65, 67, 68, 74–78
**NAT** non-autoregressive neural machine translation. 2, 3, 30, 31, 33, 36, 39, 40, 45, 47, 48, 51, 56, 57, 61, 62, 64, 66, 67, 73, 74
**NAT-REG** non-autoregressive neural machine translation with auxiliary regularization. 39, 48
**NLP** natural language processing. 5, 7, 9
**NMT** neural machine translation. 2, 3, 5–7, 9, 10, 12, 18, 19, 24, 29–32, 43, 68
**NN** neural network. 27
**NPD** noisy parallel decoding. 33, 34, 48, 57, 59, 62

**OaXE** order-agnostic cross-entropy. 38, 48
**ODD** optimal deduplicated decoding. 45
**OOV** out-of-vocabulary. 7, 9

**RNN** recurrent neural network. 2, 6, 9, 10, 12, 13, 17, 26, 31, 32, 42, 52

**SAN** self-attentive network. 6
**SGD** stochastic gradient descent. 21, 22
**SMART** semi-autoregressive training. 42

**TPU** tensor processing unit. 24

**WMT** Conference on Machine Translation. 26, 27, 46, 48, 50, 56, 59, 63, 64, 67–70, 72–77
**WNGT** Workshop on Neural Generation and Translation. 29

**XLM** cross-lingual language model. 41, 42

# List of Tables

# List of Figures

# List of Publications

Helcl, J. – Libovický, J. Neural Monkey: An Open-source Tool for Sequence Learning. *The Prague Bulletin of Mathematical Linguistics*. Apr 2017, 107, 1, p. 5–17. ISSN 0032-6585

- This paper introduces a software tool *Neural Monkey* which was used for the experiments in Chapter 4 of this thesis.
- Citations (without self-citations): 54

Libovický, J. – Helcl, J. Attention Strategies for Multi-Source Sequence-to-Sequence Learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, p. 196–202, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-2031. Available at: https://aclanthology.org/P17-2031

- In this paper we introduce techniques for combining multiple different inputs in sequence-to-sequence learning using RNNs.
- Selected as an Outstanding Paper at ACL 2017.
- Citations (without self-citations): 140

Miceli Barone, A. V. – Helcl, J. – Sennrich, R. – Haddow, B. – Birch, A. Deep architectures for Neural Machine Translation. In *Proceedings of the Second Conference on Machine Translation*, p. 99–107, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4710. Available at: https://aclanthology.org/W17-4710

- This paper explores deep architectures of RNN-based neural networks for MT.
- Citations (without self-citations): 80

Helcl, J. – Libovický, J. – Kocmi, T. – Musil, T. – Cífka, O. – Variš, D. – Bojar, O. Neural Monkey: The Current State and Beyond. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*, p. 168–176, Boston, MA, March 2018. Association for Machine Translation in the Americas. Available at: https://aclanthology.org/W18-1816

- A paper summarizing the Neural Monkey software at the beginning of 2018.
- Citations (without self-citations): 7

Libovický, J. – Helcl, J. – Mareček, D. Input Combination Strategies for Multi-Source Transformer Decoder. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, p. 253–260, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-6326. Available at: https://aclanthology.org/W18-6326

- The paper introduces different strategies for input combination in sequence-to-sequence models with self-attentive encoder and decoder.
- Citations (without self-citations): 43

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*, p. 3016–3021, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1336. Available at: https://aclanthology.org/D18-1336

- This paper introduces the novel CTC-based approach for non-autoregressive neural machine translation, described in Chapter 4 and extended in Chapter 5.
- Citations (without self-citations): 67

Chen, P. – Helcl, J. – Germann, U. – Burchell, L. – Bogoychev, N. – Barone, A. V. M. – Waldendorf, J. – Birch, A. – Heafield, K. The University of Edinburgh's English-German and English-Hausa Submissions to the WMT21 News Translation Task. In *Proceedings of the Sixth Conference on Machine Translation*. Association for Computational Linguistics, 2021

- This system description presents the University of Edinburgh's submissions to the WMT 21 News Translation Shared Task. The teacher models described in this paper were used in Chapter 5.
- Citations (without self-citations): 0

Only publications relevant to this thesis are included. The number of citations was computed using Google Scholar. Total number of citations of publications related to the topic of the thesis (without self-citations): 391