BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

EXPLORING CONTEXTUAL INFORMATION IN NEURAL MACHINE TRANSLATION

VYUŽITÍ KONTEXTU V NEURONOVÉM STROJOVÉM PŘEKLADU

TERM PROJECT
SEMESTRÁLNÍ PROJEKT

AUTHOR JOSEF JON
AUTOR PRÁCE

SUPERVISOR doc. RNDr. PAVEL SMRŽ, Ph.D.
VEDOUCÍ PRÁCE

BRNO 2018

Abstract

This works explores means of utilizing extra-sentential context in neural machine translation (NMT). Traditionally, NMT systems translate one source sentence into one target sentence, without any notion of the surrounding text. This is clearly insufficient and different from how humans translate text. For many high-resource language pairs, translations produced by NMT may be under certain, strict conditions, nearly indistinguishable from human produced translations. One of these conditions is that evaluators score the sentences separately. When evaluating whole documents, even the best NMT systems still fall short of human translators. This motivates the research of employing document level context in NMT, since there might not be much more space left to improve translations on the sentence level, at least for high resource languages and domains. This work summarizes recent state-of-the art approaches to context utilization, implements several of them, evaluates them both in terms of general translation quality and on specific context related phenomena, and analyzes their advantages and shortcomings. A hand-made context phenomena test set for English to Czech translation was created for this task.

Abstrakt

Tato práce se zabývá zapojením mezivětného kontextu v neuronovém strojovém překladu (NMT). Dnešní běžné NMT systémy překládají jednu zdrojovou větu na jednu cílovou větu, bez jakéhokoliv ohledu na okolní text. Tento přístup je nedostačující a neodpovídá způsobu práce lidských překladatelů. Pro mnoho jazykových párů je dnes za splnění určitých (přísných) podmínek výstup NMT nerozeznatelný od lidského překladu. Jedna z těchto podmínek je, že hodnotitelé skórují přeložené věty nezávisle, bez znalosti kontextu. Při hodnocení celých dokumentů je výstup NMT stále hodnocen hůře, než lidský překlad, i v případech, kdy byl na úrovni jednotlivých vět preferován. Tato zjištění jsou motivací pro výzkum zapojení kontextu na úrovni dokumentu v NMT, je totiž možné, že na úrovni vět již není mnoho prostoru ke zlepšení, alespoň pro jazykové páry a domény bohaté na trénovací data. Tato práce shrnuje současné přístupy zapojení kontextu do překladu, několik z nich je implementováno a vyhodnoceno v rámci obecné překladové kvality i na překladu specifických fenoménů souvisejících s kontextem. Pro zhodnocení kvality jednotlivých systému byla ručně vytvořena testovací sada pro překlad z anglického do českého jazyka.

Keywords

NMT, neural machine translation, context, recurrent neural networks, transformer, document level translation, discourse, cohesion, coherence

Klíčová slova

NMT, neuronový strojový překlad, kontext, rekurentní neuronové sítě, transformer, strojový překlad na úrovni dokumentů, diskurz

Reference

JON, Josef. *Exploring Contextual Information in Neural Machine Translation.* Brno, 2018. Term project. Brno University of Technology, Faculty of Information Technology. Supervisor doc. RNDr. Pavel Smrž, Ph.D.

Exploring Contextual Information in Neural Machine Translation

Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. RNDr. Pavla Smrže Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . .
Josef Jon
May 22, 2019

Acknowledgements

I would like to express my thanks to my advisor Doc. RNDr. Pavel Smrž, Ph.D. and to Ing. Martin Fajčík who provided me with advice and guidance, and to Catarina Cruz Silva, with whom we started a related project at Machine Translation Marathon 2018. I would also like to thank my girlfriend, Kristýna, who provided me with food and support during my work.

Contents

Department of Computer Graphics and Multimedia (DCGM) Academic year 2018/2019

Master's Thesis Specification

21979

Student:	**Jon Josef, Bc.**
Programme:	Information Technology Field of study: Information Systems
Title:	**Exploring Contextual Information in Neural Machine Translation**
Category:	Speech and Natural Language Processing

Assignment:

1. Get acquainted with current methods employing context in neural machine translation (NMT).
2. Prepare English-Czech corpora for training and evaluating NMT models utilizing context.
3. Implement, train, and compare most promising current approaches dealing with context information in NMT.
4. Design and implement extensions addressing shortcomings of the current methods or their evaluation.
5. Create a poster presenting your work, its goals and results.

Recommended literature:

- Rachel Bawden, Rico Sennrich, Alexandra Birch, Barry Haddow (2018). Evaluating Discourse Phenomena in Neural Machine Translation
- Agrawal, R., Turchi, M., and Negri, M. (2018). Contextual Handling in Neural Machine Translation: Look Behind, Ahead and on Both Sides.
- Elena Voita, Pavel Serdyukov, Ivan Titov, Rico Sennrich (2018). Context-Aware Neural Machine Translation Learns Anaphora Resolution.
- Lesly Miculicich, Dhananjay Ram, Nikolaos Pappas, James Henderson (2018). Document-Level Neural Machine Translation with Hierarchical Attention Networks.
- Sameen Maruf, Gholamreza Haffari (2018). Document Context Neural Machine Translation with Memory Networks.

Requirements for the semestral defence:

- Functional prototype of the system

Detailed formal requirements can be found at http://www.fit.vutbr.cz/info/szz/

Supervisor:	**Smrž Pavel, doc. RNDr., Ph.D.**
Head of Department:	Černocký Jan, doc. Dr. Ing.
Beginning of work:	November 1, 2018
Submission deadline:	May 22, 2019
Approval date:	November 6, 2018

Chapter 1

Introduction

Quality of state-of-the-art machine translation systems has improved vastly over the last few years, thanks to shifting the paradigm from phrase-based statistical machine translation to models based on complex artificial neural networks.

In 1986, Martin Kay [27] stated reasons why high quality machine translation is not possible – but that was before the „statistical revolution“ [22], in times of rule-based systems and symbolic AI. Nowadays, there is almost no doubt that high quality machine translation is feasible in some conditions – in several test scenarios, recent neural machine translation (NMT) systems are evaluated on par with, or even better than human translators. However, challenges mentioned in Kay's statement, and several others, still hold true today and are not addressed even in the current state-of-the-art. This work is focused on one of these challenges – utilizing discourse-level, cross-sentence context in NMT. Current systems usually use only one sentence as their input, which is clearly insufficient, as a single sentence may not contain enough information for a proper translation of itself. Context necessary for correct translation of a sentence can often lie outside the sentence, and this fact makes it impossible to translate such sentence correctly even for a perfect MT system, if that system adheres to one-to-one sentence paradigm.

Exploiting the discourse addresses many interesting sub-problems, like adaptation to topic, genre, domain, or author's style, discourse consistency (e.g. lexical consistency - using the same translation for one entity throughout the whole document) or coherence and cohesion, which includes coreference resolution (e.g. cross-lingual pronoun disambiguation, also mentioned in Kay's paper [27]).

However, utilizing context is more than solving each of the problems mentioned above separately, since discourse can contain information that is not contained in any of the sentences alone, as described later on. As stated by Kehler [26]: „The meaning of a discourse is greater than the sum of the meanings of its parts.“. For this reason, my work focuses on end-to-end learning, using neural networks and raw training data without any linguistic preprocessing (like coreference resolution), rather than developing specialized tools designed to tackle these phenomena separately.

In the second chapter, I describe phenomena that this work addresses from a theoretical point of view, and then I summarize publications related to my work. Since employing context in NMT is only just getting into focus of researchers, this chapter is rather brief and mostly comprises of analyzing papers that were published during the last year.

In the next chapter, issues concerning data preparation are discussed, including obtaining corpora with document-level information, preproccessing them and creating evaluation data and protocols. Since the effect of discourse level phenomena on sentence level quality metrics is hard to predict, I evaluate the systems with test sets and metrics designed specifically for this task, aside from general translation quality measurements.

In the fourth chapter, several context-aware architectures, inspired by a recent related work, are proposed and described. Their theoretical advantages and shortcomings are discussed. The fifth chapter describes implementation of the proposed models in a C++ NMT framework Marian [25].

The next chapter describes the central part of this work. Architectures described in previous two chapters are evaluated on test sets described in the third chapter. I analyze their shortcomings and determine which approach performs the best. Issues regarding machine translation evaluation are also discussed in this chapter.

Finally, in the last chapter, I draw conclusions from the experiments and plan future work.

In summary, I implemented some of the recent techniques of utilizing context in NMT and I evaluated them in terms of both general translation quality, and accuracy on translation of specific discourse phenomena. A specialized English-Czech test set was created for this purpose.

Chapter 2

Machine translation and context

This work focuses on translation of a text given a document context or discourse. In first part of this chapter, discourse phenomena are described from a theoretical point of view, focusing more on the linguistic and less on the engineering side. First, a summary of what a discourse is and what phenomena are considered as being part of discourse is presented. Some of these phenomena are described more in detail, as they are important later in this work. In the second part, principles of current state-of-the-art NMT models are discussed. In the final part of this chapter, several papers related to this work are analyzed.

2.1 Discourse

Since this work focuses on translation of sentences utilizing a document context, or discourse (these terms will be used interchangeably), it may be beneficial to define what *discourse* means. Eisenstein [13] characterizes discourse simply as "multi-sentence linguistic phenomena“ in his recent NLP notes. Andrew Kehler [26] refers to discourse as "collocated, related groups of sentences“. Kendall and Wickham [28] say that a discourse is a corpus of statements whose organization is regular and systematic. Broader definition of discourse is that it is the use of spoken or written language in context of society. For the rest of this work, it will be assumed that a discourse means multiple sentences, which have some kind of connection between each other. An interesting point is that the meaning communicated by the discourse is bigger than the sum of meanings of individual sentences. Discourse can contain information that none of the sentences contains by itself.

2.1.1 Cohesion, Coherence, Consistency

According to [17], discourse is mainly embodied in three aspects: cohesion, coherence and consistency. These terms are used in some of the related papers described later, so it may be useful to define them at least in general terms. Generally, cohesion is a surface property of text that qualifies whether the sentences are correctly linked to each other, using special words, also called *cohesive devices* [16]. In other words, *cohesive devices* are linguistic units that are used to tie parts of discourse together, e.g. expressions like *however*, *although* or *on the other hand*. In contrast, coherence concerns abstract, mental image of meaning of the text as a whole. Coherent text should be comprehensible and not confusing for the reader. Linguists are not united in their view on cohesion and coherence definitions and relationship. For example, Mey [43] defines the relationship as follows: „cohesion establishes local relations between syntactic items (reference, concord and the like), whereas coherence has to do with the global meaning involved in what we want to express through our speech activity.“ Summary of different views on coherence and cohesion can be found in a work by Schmeid et al. [53]. Effort to analyze a subcategory of cohesion phenomena is called coreference resolution.

2.1.2 Coreference resolution

Several concepts regarding coreference resolution are introduced in this section, since coreference resolution is tightly related to a context-aware translation, and it can be used to evaluate a translation system's ability to exploit context information.

Reference is a relationship between a linguistic expression and the object or idea represented by the expression. Given a sentence:

Kristýna *went to a grocery shop, but* ***she*** *realized* ***she*** *left the shopping list home.*

In this sentence, *Kristýna* and both occurrences of *she* denote a person called Kristýna. These expressions are called *referring expressions* [13] or *mentions* [26] and the entity they refer to (person named Kristýna) is called the *referent*. When multiple referring expressions are referring to the same referent, they *corefer*. All previous coreferences of a mention are called *antecedents*. If an entity that was defined earlier in text is referenced, as is the case in the example sentence, this reference is called *anaphoric*. Inversely, *cataphora* is the term for the case when the referent is mentioned after the referring expression, as in the following two sentences:

Who was ***that****? That was* ***Bedřich****, an old friend of mine.*

The goal of coreference resolution is to find which referring expressions in a text refer to the same referent. This is a very challenging task, since apart from linguistic reasoning, effective algorithm would need to contain real world knowledge and be able to do a common sense reasoning. In fact, there is a Turing test inspired challenge to determine whether an algorithm is a true general artificial intelligence based on coreference resolution – the Winograd Scheme Challenge[1].

A special, well-studied case of anaphoric expressions are pronomial anaphora. Resolution of pronomial anaphora and a correct translation of a pronoun with antecedent outside the sentence is used as an evaluation metric in several papers presented in the related work section. Another way of exploiting coreference resolution for NMT evaluation is to look at attention matrix for a word with an antecedent in the context sentence and see if the word wich is most attended to agrees with a human judgement (or a coreference resolution system) of the atencedent. Such approach was used by Voita et al. [69] and it is further disscussed in section Related work.

2.2 Machine translation

Machine translation (MT) is, simply put, translation of text in one language into another language using a computer program. First MT experiments were carried out in 1950s, including the Georgetown–IBM experiment, which was designed to attract funding for the new

[1] http://commonsensereasoning.org/winograd.html

research area. The experiment consisted of translating 60 sentences (carefully picked beforehand) from Russian into English. Using 6 grammatical rules and vocabulary of 250 lexical terms the researchers were able to successfully demonstrate the system in operation. Both the rules and the vocabulary were crafted to the beforehand known sentences. This demonstration, along with statements claiming that MT will be a solved problem in next few years, led to increased funding for MT projects.

One of the early MT researchers, Yehoshua Bar-Hillel, argued that in order to be able to achieve *fully automatic, high quality translation* (FAHQT), machines would need to have world knowledge and understand the meaning behind the input text. For that reason, Bar-Hillel argued that FAHQT should not be the ultimate goal of MT research, but the researchers should instead focus on systems providing help to human translators, thus speeding up the translation process [4]. He criticized his colleagues for not coming to terms with this fact, rising false hopes among investors and governments, and waiting for a FAHQT system that was never to come. His view was proven right by the Automatic Language Processing Advisory Committee (ALPAC) report in 1966. The report was very sceptical of current state and prospects of MT, advised both the public and the researchers to lower their expectations, and finally led to significant decrease of MT funding by the U.S. government and companies.

In the next two decades, MT had some limited successes in specific applications. Systems in that epoch generally analyzed semantic, syntactic and morphological aspects of the input text, and used rules and bilingual vocabularies to translate it into the target language. This approach to MT is called *rule based machine translation* (**RBMT**). One of the successful systems, named METEO, was used by the Canadian government to translate weather forecasts between English and French [35].

A significant breakthrough came in the late 1980s. In 1988, researchers in IBM laid foundation for a new approach to the topic, so called *statistical machine translation* (**SMT**) [8]. SMT uses models trained on a large amount of parallel corpora (meaning same documents in two languages, where sentences and their translations are aligned) to estimate a probability that a string in a target language is a translation of some other string in a source language. This probability estimation is usually based on Bayes rule and heuristics to limit the search space. First types of SMT systems were word-based models, in which the unit of translations were single words, translated into number of target words (including none), in other words, these systems align words in source and target sentences. Such system is clearly very constrained, but the obtained word alignments can be used as a training data for a more recent and advanced category of SMT systems, called *phrase based statistical machine translation* (**PBSMT**).

Most notable example of these systems is Moses [32], an open source project by Philipp Koehn and many others. Moses and similar frameworks address many shortcomings of word based SMT by using a phrase as an atomic unit, instead of a word. This allows them to make use of local context and perform many-to-many translations. These phrases are not necessarily phrases in linguistic sense, rather they are n-grams of words based on statistics of their occurrences in the training data. The phrase model is usually coupled with another models that perform auxiliary analysis of the text and influence the translation. The most usual ones are reordering model, which changes the order of phrases in translation, and language model, which estimates the probability of candidate sentence translation in the target language (not conditioned on the source sentence, in other words, helps to create more natural and fluent translations).

PBSMT marked big leap in quality of translation for language pairs with big quantities of parallel data and made web translation services, such as Google Translate, possible and practical. This approach was state-of-the-art in machine translation up until 2014.

2.2.1 Neural machine translation

In 2014, two papers with major impact on MT were released by Sutskever et al. [63] and Cho et al. [9]. The main differences compared with PBSMT were the following two. First, the NMT systems are using continuous, distributed representation of words. Such representations came into focus of researches few years earlier, for example see paper by Mikolov et al. [44]. This means that words that appear in similar contexts in the training data, are processed similarly by the model, and that the representation is more semantic, in the spirit of J. R. Firth's quote:

> *You shall know a word by the company it keeps.*

Second big shift from PBSMT is that the NMT systems use one model, based on encoder-decoder neural network, performing all the necessary operations, instead of combination of separate models engineered for each task (e.g. translation, reordering, language modeling, as mentioned earlier). The network performs transformation of one sequence (source sentence) into another sequence (target sentence). Such networks are collectively called sequence-to-sequence models. More specifically, encoder reads the input sequence (source sentence), converts the input words into continuous representations (vectors) using an embedding layer, and based on these representations, computes a representation of the whole sentence. Based on this representation, decoder iteratively, word-by-word, generates the translation. Already generated words form a second input of the decoder.

Nowadays, one of two types of deep neural networks are used in practice. First, recurrent neural networks (RNN), which were the first ones being used for MT, along with convolutional networks. These networks usually use either LSTM or GRU units, and are further improved by an attention mechanism [2]. Since Summer 2017, RNNs are being replaced by self-attention based models [68], which are more parallelizable, since they remove the need for sequential processing of the input sequence inside the network, and also usually offer superior translation quality. Both architectures are described in more detail in the following two sections.

2.2.2 Recurrent neural networks in machine translation

Recurrent neural networks (RNN) were used in both of the fundamental NMT papers mentioned above. Almost simultaneously in 2014, Cho et al. [9] and Sutskever et al. [63] published neural network based MT systems and results that led to an increase of research in this direction. Both of the systems used encoder-decoder RNN networks with LSTM or GRU units. Both networks reached impressive results, achieving comparable performance with strong PBSMT baseline system in terms of the BLEU [47]. Cho and his colleagues performed analysis of relationship between sentence length and BLEU, presented in Figure 2.1.

As apparent from the figure, the performance of the system degrades rapidly for longer sentences, probably because only one fixed-sized vector is used to compress all the information about the sentence in the encoder. In the subsequent work, Bahdanau, Cho, and Bengio [2] proposed a solution - an attention mechanism which allows the decoder to look at all of the encoder states, not only the last one. Such network became a de facto baseline

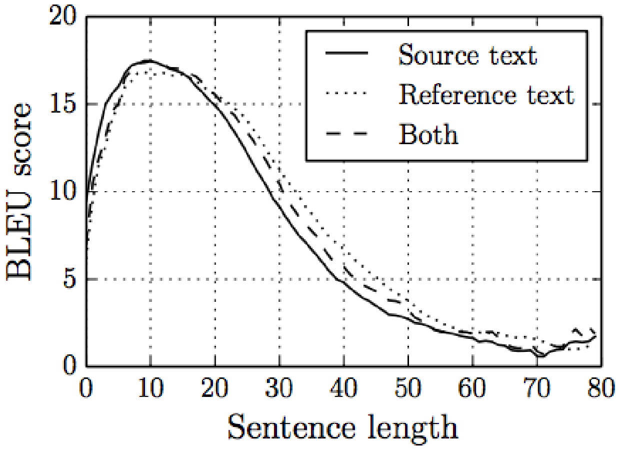


Figure 2.1: Relationship between sentence lengths and BLEU scores shows that the original RNN encoder-decoder architecture had difficulties with longer sentences. Based on this analysis, an attention mechanism between encoder and decoder was presented. Figure taken from [9]

.

for NMT for the next three years, with many extensions and improvements built around this general architecture. A high level overview of encoder-decoder network with attention used in production (Google NMT system [72]) is presented in Figure 2.2.

2.2.3 Transformer

In Summer 2017, Google AI presented a paper [68] describing a novel NMT architecture, addressing many of the RNNs shortcomings. Most notably, the new architecture removes the need for sequential (recurrent) processing of the input sentence, which is responsible for much of the sequential computing of the RNN network, thus speeding up the training on modern GPUs, which are vastly parallel.

Instead of LSTM or GRU recurrent units, the Transformer encoder utilizes a mechanism called self-attention, which models a relationship between all the input words and computes a representation of a word by comparing it with all the other words in the sentence. This comparison produces an attention score, which determines how much should the representation of the other words contribute to representation of the current word. This procedure is repeated in parallel for all the input words for a constant (empirically chosen, 6 for the base model) number of steps, i.e. layers.

Processing all the words at once allows the network to capture long-term dependencies better than RNNs are able to, since distance of the words in the sentence does not matter. RNNs have problems with long input sequences, even when gating and attention mechanisms are employed. The root cause of this is the inherently sequential nature of RNN processing – input symbols are processed one by one, the number of operations between processing two input symbols is linear in regard to their distance in the input sequence. During each operation on the path from first symbol to a second symbol, some part of the information about the first symbol is not propagated further. Thus, information about the first symbol is incomplete when creating representation of the second symbol. This is

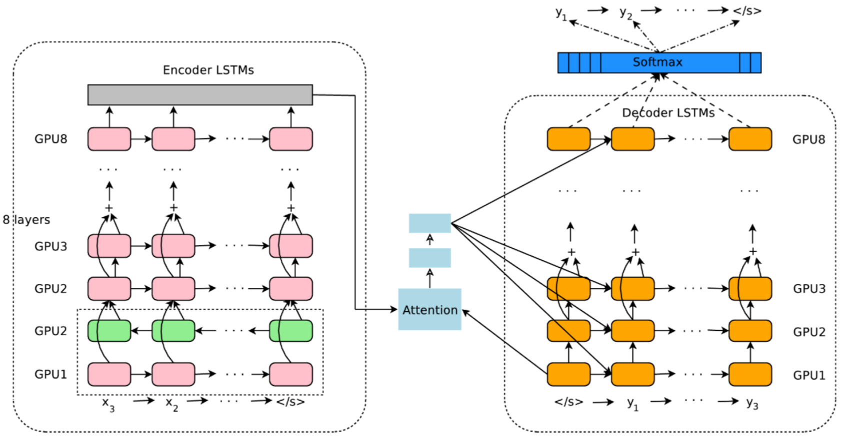


Figure 2.2: Google's neural machine translation. Taken from [72]

.

not the case with the Transformer – the self-attention mechanism connects all the symbol representations directly. This notion is further discussed in Chapters 4 and 6.

An issue with processing all input symbols at once is that the model does not have any information about the relative positions of the symbols. This information needs to be added explicitly. The position of the word in the sentence is encoded by a sinusoidal function and concatenated with the word representation.

The decoder works in a similar fashion to the encoder, only difference being that it also attends to previously generated words, in addition to input words. Final layer of the decoder performs the softmax function with dimensionality equal to size of a target vocabulary. Output of this softmax is a probability distribution over words in the target vocabulary and the generated output word is selected based on this probability. Overview of the architecture can be seen in Figure 2.3. Transformer networks outperform RNNs in most applications (for example see paper by Lakew et al. [33]), one of the disadvantages is their sensitivity to hyperparameters, like learning rate schedule or minibatch size, see papers by Popel and Bojar [49] or Ott et al. [46].

Model description

Since the experimental architectures described later in this work are modifications of the Transformer, the original model is described in detail in this section, following description in the original paper [68]. The architecture follows the general structure of encoder and decoder. Encoder creates a vector representations of the input sequence. Let the input symbols be denoted $(x_1, ..., x_n)$ and the continuous representations generated by the encoder $\mathbf{z} = (z_1, ..., z_n)$. The decoder sequentially generates output symbols $(y_1, ..., y_m)$, based on $\mathbf{z}$ and symbols generated in previous steps.

Encoder The encoder consists of six identical layers, each one formed by two functional blocks or sublayers – multi-head self-attention followed by position-wise, fully connected feed forward network. A residual connection [19] is built around each of the functional blocks, followed by layer normalization [36]. This results in a final output function in form $LayerNorm(x + Sublayer(x))$, $Sublayer(x)$ being the function performed by the sublayer itself.

The first layer of encoder uses word embeddings as an input, i.e. list of vectors (512-dimensional for the base model) generated by the embedding layer for each token in the source sentence. Length of the list is the maximum sentence length, which can be set as a hyperparameter. Subsequent layers use outputs of the previous layer as an input.

Decoder The decoder is again built from six identical layers, but in contrast to the two sublayers in the encoder, each layer consists of three sublayers. A multi-headed encoder-decoder attention layer is inserted between the self-attention and feed-forward layers, which are the same as in the encoder. Again, residual connections and layer normalization are used. The decoder generates output symbols step by step. Additionally, the self-attention in the decoder is masked, so that it can only attend to positions already generated in the previous timesteps. In another words, the masking prevents the decoder to attend to positions that were not yed generated, so the symbol that is being predicted depends only on the encoder representations and previously generated symbols.

Attention The authors define **attention** as a function of three variables (vectors): query **Q**, key **K** and value **V**. A weight of each element in the value **V** is computed by a compatibility function of query **Q** and key **K**. Attention output can be then computed using these weights as a weighted sum of elements in **V**.

In the original paper, the particular implementation of attention function is called *scaled dot-product attention*. The function has three vectors as parameters, **Q** and **K** of dimensionality d_k and values **V** with d_v dimensions. The attention function output matrix is then computed using the following formula:

$$Attention(Q, K, V) = \operatorname{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{2.1}$$

The dot product of a query and all keys (i.e. the compatibility function) is computed first, then divided by square root of dimensionality of the key vector. Softmax function is then applied to obtain the weights for corresponding values.

The attention used in Transformer is **multi-headed**, meaning that instead of computing one set of attention weights (with dimension d_{model}), h different linear projections of queries, keys and values are learned. The **Q**, **K** and **V** vectors are projected to $d_k = d_v = d_{model}/h$ dimensions, in case of the original model to $512/8 = 64$-dimensional vector. The attention function described above is performed on each of these vectors in parallel and the d_v-dimensional outputs are concatenated and projected again, resulting in the final output of the multi-head attention layer.

$$\begin{aligned} \text{MultiHead }(Q, K, V) &= \text{ Concat (head }_1, \ldots, \text{ head }_\mathrm{h})W^O \\ \text{where head }_i &= \text{ Attention }\left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{2.2}$$

Where $W_i^Q \in \mathbb{R}^{d\text{ model } \times d_k}, W_i^K \in \mathbb{R}^{d_\text{model} \times d_k}, W_i^V \in \mathbb{R}^{d_\text{model} \times d_v}$ are the learned input projection matrices for each of the heads, and $W^O \in \mathbb{R}^{hd_v \times d_\text{model}}$ is the weight matrix of the final projection after the concatenation of the outputs of each of the heads.

The multi-head attention is used in three contexts in the Transformer. Firstly, in the **encoder self-attention**, **Q**, **K** and **V** are all the same vector, obtained as an output of the previous layer (or embedding layer in case of the first layer). Each position can attend to representations for all the other positions from the previous layer. In **decoder self-attention**, the mechanism works similarly, only difference being masking. Each position can only attend to position to the left of itself, which is implemented by setting values of representations that break this condition to $-\infty$ before the softmax computation. This ensures that the decoder only looks at previously generated symbols. Finally, **encoder-decoder attention** is used similarly as in other encoder-decoder models. The queries are obtained from previous decoder layers, while the key-value pairs are representations from the last layer of the encoder. Each position in decoder can attend to all of the encoder representations.

Position-wise feed-forward layer The second type of sub-layer used in the Transformer is a feed-forward fully connected network. This network is applied to each position separately and consists of a linear transformation, followed by a ReLU activation and a second linear transformation:

$$\mathrm{FFN}(x) = \mathrm{ReLU}\left(xW_1 + b_1\right)W_2 + b_2 \tag{2.3}$$

Embeddings As usual in other NMT models, the Transformer learns embeddings that map input and output tokens to a d_{model}-dimensional vector representation. To convert the decoder output into a token probability, linear transformation followed by a softmax is used. Weights of this linear transformation can also be considered a form of embedding, meaning that the model uses embeddings at three different stages of computation (input and output embeddings and the weights of the linear layer before softmax). All of these three embeddings share the same weights – there is only one embedding weight matrix.

Positional embeddings One of the most notable improvements of the Transformer over RNN models is that the input is processed in parallel, without a recurrent (sequential) computation. However, this means that the model does not have any notion of the order of the input symbols. In RNNs it is encoded implicitly by the order in which the input symbols are processed. For the Transformer to be able to utilize the knowledge of positions of input tokens in the sequence, the position information must be added in some way. To achieve this, position of each token is encoded by a function and the resulting vector is summed up with the corresponding token embedding. The encoding function can be either learned, or manually designed. In the paper, authors use a hand-crafted combination of sine and cosine functions:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos/10000^{2i/d_{\text{model}}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(pos/10000^{2i/d_{\text{model}}}\right) \end{aligned} \tag{2.4}$$

Variable *pos* denotes the position of the token in the input, i is the dimension in the embedding vector. The periodicity of this function allows the model to extrapolate to lengths not seen during the training (in contrast to learned positional embeddings).

Training The author train the model using **Adam** optimizer with hyperparameters $\beta_1 = 0.9, \beta_2 = 0.98$ and $\epsilon = 10^{-9}$. Learning rate is scheduled using the following formula:

$$lr = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num.warmup\_steps^{-1.5}) \tag{2.5}$$

Learning rate is linearly increased for *warmup_steps* training steps, and then, after reaching its maximum value, it is decreased with inverse square root of the step number. Without this variation of the learning rate, the training diverges. The authors trained the models on 8 P100 GPUs, with batch sizes around 25000 tokens. In the Transformer, the batch size is important for stability of training and performance [49].

Three regularization techniques are used:

- Pervasive dropout [61] with probability $P_d = 0.1$ at the output of each sublayer, before layer normalization and summing the output with the input (residual connections).
- Dropout in the embbeding layer, after the token embeddings are summed with the positional embeddings.
- Label smoothing [64] with value 0.1.

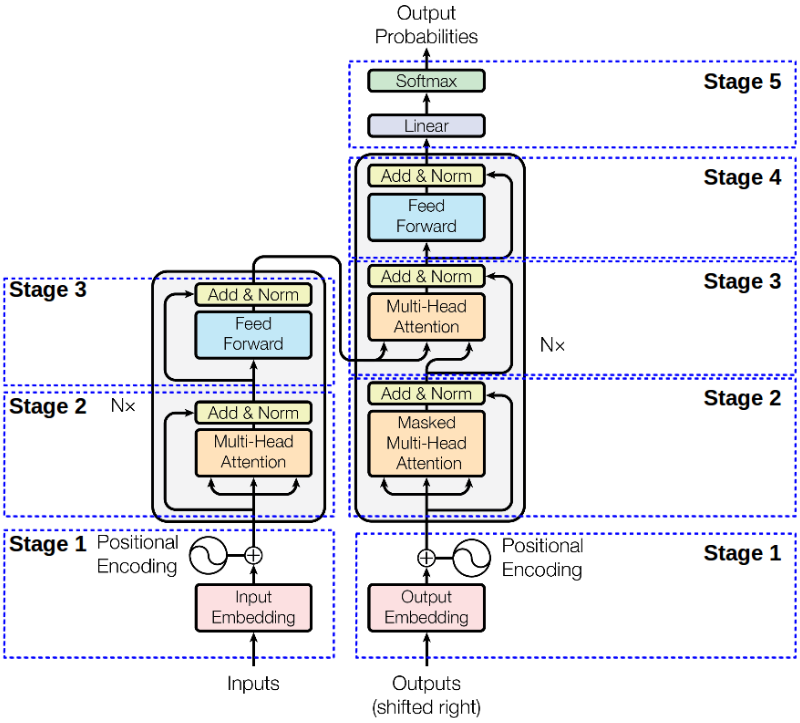


Figure 2.3: Encoder (left) and decoder (right) of the transformer model. Taken from [68]
.

The Transformer presented in this paper obtained state-of-the-art results on WMT 2014 English-German and English-French datasets, while allowing much better parallelization and thus shorter training times than its RNNs or convolutional counterparts. Transformer's superior performance in machine translation was confirmed by many subsequent papers, e.g. Lakew et al. [33]. Transformer-based models are used in many other fields of NLP, also improving state-of-the-art results [12] [52].

Other explanations Even though the original paper is clearly written, several very nicely done explanations and guides to the Transformer model can be found on the Internet. It would be a shame not to name few of them, since they might be useful to get at least a gist of how the model operates. The original Google blogpost about the Transformer[2] contains

[2] https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

high-level, intuitive and animated explanation of how and why self-attention works. The Illustrated Transformer[3], article by Jay Alammar, approaches the model from top-down perspective, with helpful figures for every level of abstraction. One of the illustrations from this post, which displays the whole model, is presented in Figure 2.4. The Annotated Transformer[4] takes the original paper and adds code and comments to the corresponding parts. The whole article is a functional implementation of the Transformer in a Jupyter notebook, using OpenNMT and Pytorch. Another similar post, with code snippets and figures, was created by Michał Chromiak[5].

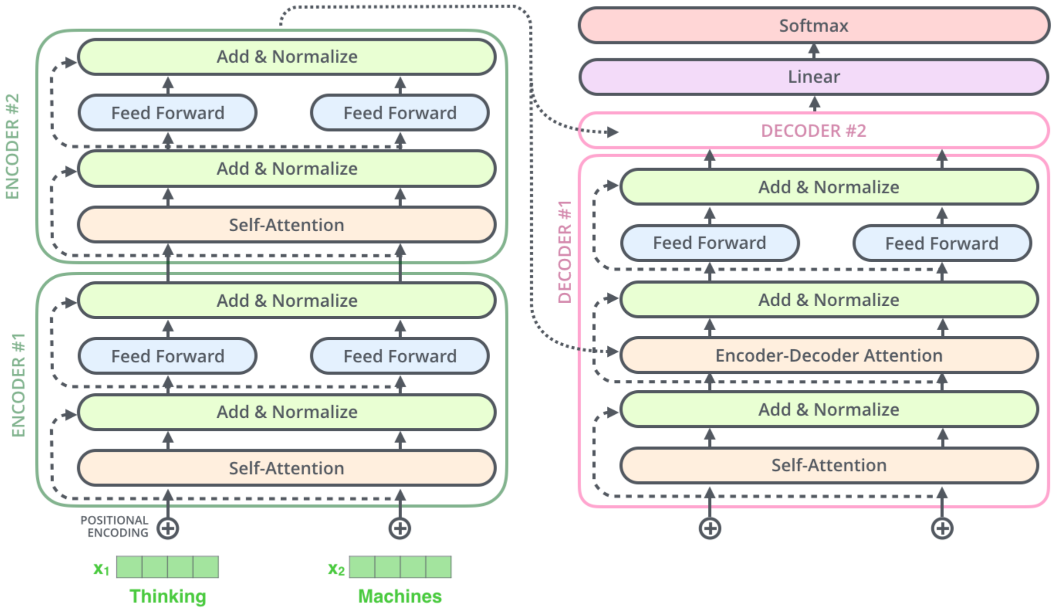


Figure 2.4: An alternative overview of the Transformer model (with 2-layer encoder and decoder) by Jay Alammar. Taken from The Illustrated Transformer (http://jalammar.github.io/illustrated-transformer/.

2.3 Related work

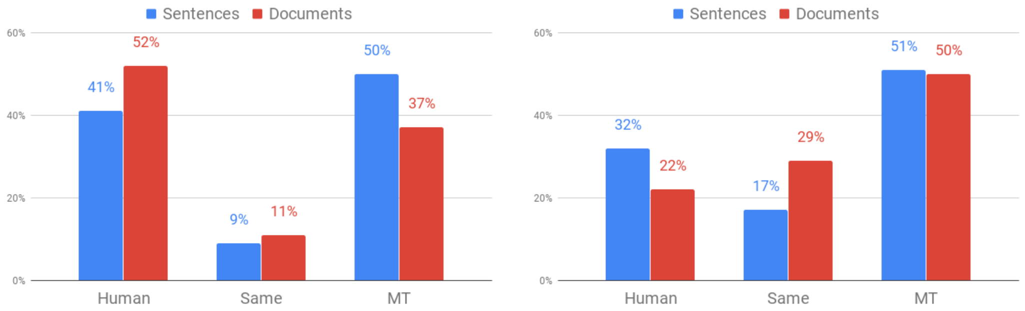
This thesis deals with employing extra-sentential context in NMT. Many publications about this topic emerged in the last two years.

After Microsoft claimed reaching human parity in Chinese-English news translation [18], **Läubli et al.** [34] analyzed these claims to asses if they are true. The authors have changed the evaluation protocol slightly: evaluators were professional translators as opposed to crowd sourced bilingual speakers used by Microsoft, and pairwise ranking was adopted instead of direct assessment. The translations were evaluated in terms of fluency and adequacy. The evaluators were shown a source sentence (in case of adequacy evaluation, fluency evaluators were only shown the two translations) and two translations, one produced

[3] http://jalammar.github.io/illustrated-transformer/

[4] http://nlp.seas.harvard.edu/2018/04/03/attention.html

[5] https://mchromiak.github.io/articles/2017/Sep/12/Transformer-Attention-is-all-you-need/#.XMzJK99fjM0



(a) Adequacy

(b) Fluency

Figure 2.5: Results of human evaluation of Microsft's English to Chinese MT system translation adequacy and fluency by Läubli et al. [34]. Increased preference of human translations can be observed in document-level evaluation.

by a human (professional translator) and one by Microsoft's MT system. They were asked two questions:

> *Which translation expresses the meaning of the source text more adequately?* (adequacy)

and

> *Which text is better English?* (fluency)

The results did in fact confirm Microsoft's claims (see Figure 2.5). In terms of adequacy, the evaluators preferred MT in 50% of the sentences, didn't have any preference in 9% and preferred the human translation in 41% of the cases. Interestingly, in terms of fluency (monolingual evaluators), the MT system output was preferred in only 32% of the examples (51% human, 17% tied) – even though fluency, and not adequacy is often cited as the main advantage of NMT.

However, when the evaluators were asked to compare whole documents, the results changed drastically – 52% for human, 11% tied , 32% MT in adequacy, 50% human, 29% tied, 22% MT in fluency. These results convincingly show the need for document level NMT.

Just for completeness, the same translations by the Microsoft's MT system were reevaluated again by [65], who found out that a large portion of the Chinese source sentences in the test set were originally English sentences, translated into Chinese. This means that a part of the data set was in fact *„translationese“*, i.e. a text seemingly in Chinese, but still retaining some of the properties of an English text. When only sentences which were originally in Chinese were considered, the MT system did not reach human parity.

In a paper by **Bawden et al.** [5], most of the RNN model architectures that incorporate context up to date are compared and evaluated on English-French dataset, which is designed to target different discourse phenomena, namely coreference, lexical cohesion and lexical disambiguation. Examples in the set are hand-crafted, but inspired by sentences found in OpenSubtitles2016, to assure similarity to real world examples.

Structure of the set is described later, in Chapter 6, as I will use it directly and also will modify it for English to Czech translation. So, only for a brief overview, in lexical disambiguation part, each source sentence has two different source side contexts (one previous sentence) and two possible translations (also with previous target sentence as a context). Each translation is correct in one of the contexts. For lexical coherence (repetition), the source context stays the same, but the previous target sentence is different for each case. The desired result is that the same source word or phrase is translated the same way in both target sentences.

Many context-aware RNN architectures were explored. The models were trained on English-French OpenSubtitles2016, and evaluated on the set mentioned above, and also in terms of general translation quality (BLEU [47], see Chapter 6 for details on this metric) on subtitle test sets from 4 genres: comedy, crime, fantasy and horror. The general translation performance depended heavily on test set used. The simplest models have ordinary encoder-decoder architecture, with concatenation of current and previous sentence on the input (separated by a special token), and either single (labeled 2-to-1) or two (previous and current, 2-to-2) sentences on the output. Generally, 2-to-2 model performed quite well – with about 0.5 BLEU gain on all subtitle test sets, 63.5 % precision on coreference and 52 % on cohesion/coherence sets. 2-to-1 scored 52 % in coreference and 53% in cohesion, and outperformed baseline in 2 out of 4 translation test sets.

Another set of experiments focused on multi-encoder models, using previous source or target sentence as an additional input. Three different attention strategies were evaluated. First of them was concatenation - the context vectors from both encoders are concatenated and a linear transformation is applied on the result to obtain vector with same size as the original vector. Second approach uses a *tanh* attention gate, which learns to give different importance to each element of the both context vectors. Finally, hierarchical attention, as in [**?**] is used - another attention layer is applied on the context vectors from each encoder. Overall, the best performing model was using one previous source sentence as an input, hierarchical attention and was trained to generate both previous and current target sentence. This model gained about +1 BLEU across all test sets and scored 72.5% on coreference and 57% on coherence/cohesion. Models from this paper are not implemented in my work, with the exception of concatenation and 2-to-1 dual encoder model, since only RNN models were explored, while my work focuses mainly on the Transformer model. However, this paper is crucial for my work because of the test set and evaluation methodology used in it.

One of the earliest attempts on incorporating discourse into NMT is a work by **Jean et al.** [21]. Presented system utilizes a dual encoder RNN, with one encoder for a source sentence, as usual, and another auxiliary encoder for a context sentence. Attention mechanism for the contextual encoder also has source vector from the main attention as an input, besides the usual inputs (previous symbol, previous decoder state, annotation vector). The models were trained on WMT16 and ISWLT En-De and En-Fr data sets, which belong roughly to news domain.

The authors evaluated their model in terms of general translation quality (BLEU), as well as in more focused evaluation - pronoun prediction (RIBES). For the pronoun prediction, pronouns in the target sentence were replaced with a special token, then sentences with all the possible combinations of pronouns in place of these tokens were generated and the one with best log-probability, as scored by the model, was chosen as the output. There were improvements observed for both of the metrics when using small training data - ISWLT or WMT16 reduced to up to about 40%. However, when the model was trained on a larger corpus, the improvements vanished. Another interesting outcome is that both baseline and contextual model outperformed all the submissions for WMT16 pronoun prediction shared task, even though these submissions were trained specifically for this task.

In a paper by **Wang et al.** [70], the authors focus on employing context from previous source sentences using hierarchical RNNs. Target context was not used, because in the preliminary experiments, the target context hurt the translation quality due to error propagation. First, the sentence-level RNN reads the sentence word by word and summarizes the content in its last hidden state. Document-level RNN uses the last hidden states of n sentence-level RNNs as an input and its last hidden state is considered a global context vector D. Authors use $n=3$ for the experiments in this paper.

Two strategies of adding context to the NMT model are explored - *Initialization* and *Auxiliary input*. *Initialization* means that encoder, decoder or both are initialized with D. In encoder, all the states are traditionally zero at the begging of the sentence translation, so D is simply used as an initial state of the encoder layers. In decoder, the usual formula for computing the hidden state in the first step is changed from $s_0 = tanh(W_s h_N)$ to $s_0 = tanh(W_s h_N + W_D D)$, where h_N is the encoder last state and W_h, W_D are trainable weights.

In the *Auxiliary context* scenario, D is used as an additional input for the decoder. In an usual NMT model, the decoder hidden state in time step i is computed as $s_i = f(s_{i-1}, y_{i-1}, c_i)$, where s_{i-1} is the previous state of the decoder, y_{i-1} is the last generated symbol, c_i is the sentence context vector from the encoder and attention mechanism. After adding the global context vector D, the formula changes to following:

$$s_i = f(s_{i-1}, y_{i-1}, c_i, D)$$

.

On implementation level, the authors simply concatenated c_i and D into a single context vector. However, the necessity of global context is different for each word, e.g. translation of ambiguous words can benefit from more context information than translation of words with no ambiguities. To address this observation, the authors used a sigmoid context gate, as in a paper by Tu et al. [66]. The gate uses the previous symbol, previous hidden state and encoder context vector to generate a vector of same dimension as D. More formally,

$$z_i = \sigma(U_z s_{i-1} + W_z y_{i-1} + C_z c_i)$$

where z_i is the gate output and U_z, W_z and C_z are learned weights.

After applying the sigmoid function, all elements of z_i are between 0 and 1 and z_i has the same size as D and intuitively, we want for each element from z_i to tell the decoder how much of corresponding element from D to use in generating the next symbol. To achieve this, D is multiplied element-wise with z_i:

$$s_i = f(s_{i-1}, y_{i-1}, c_i, z_i \otimes D)$$

Experiments were carried out on a subset of Chinese-English LDC[6] corpora that contains datasets with document boundaries. Implementation of the models is based on Nematus[55], using vanilla Nematus as a baseline (30.57 average BLEU on 3 test sets). All of the modifications had positive impact on BLEU score, and are mostly complementary - namely, using encoder initialization the network achieved 31.55 BLEU, decoder initialization 31.90 BLEU

[6] https://www.ldc.upenn.edu/

and combining both 32 BLEU. Auxiliary context strategy reached 31.3 BLEU without the gate, and 32.24 when the context gate was used. All of the improvements together yielded improvement of 2.1 BLEU over the baseline, reaching 32.67 BLEU on average.

The authors also performed a manual analysis of the errors in inter-sentence phenomena. They randomly chose part of the test set and counted number of errors in translation of ambiguous words and phrases and in consistency of the translation. The system solved 76% of ambiguity and 75% of consistency errors, while bringing in 26% of new errors (relative to the original counts).

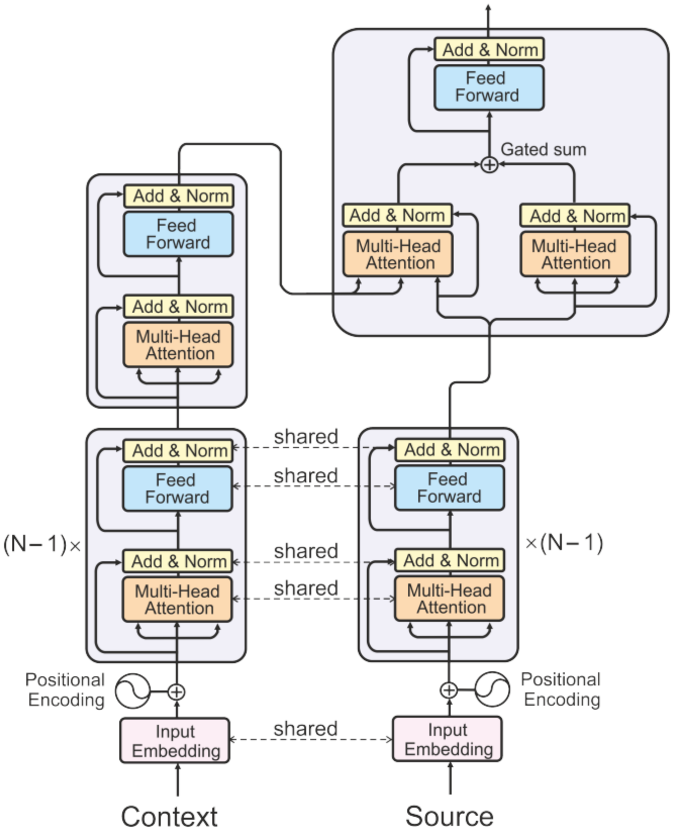


Figure 2.6: Context sentence (left) and source sentence (right) encoders of the context-aware Transformer proposed by Voita et al. [69].

Paper by **Voita et al.** [69] presents a modified Transformer model with a specialized context encoder, with some of the encoder layers weights shared between context and source encoders. In the preliminary experiments, neither concatenation nor simple dual encoder models, proposed in earlier work, worked well with the Transformer. The models were trained on OpenSubtitles corpus, and resulted in improvement in BLEU (0.7 points), pronoun disambiguation and coreference resolution.

The proposed model consists of two encoders – context and source encoder – and a decoder. Overview of the encoder part of the model is presented in Figure 2.6. The decoder is identical as in the vanilla Transformer model. More detailed description of the architecture is presented in Chapters 4 and 5, since this is one of the models implemented and evaluated in my work.

Type	Baseline	Context-aware	Difference
Masculine	26.9	27.2	+0.3
Feminine	21.8	26.6	+4.8
Neuter	22.1	24.0	+1.9
Plural	18.2	22.5	+4.3

Table 2.1: Improvements in BLEU scores for test sets with a pronoun „it“ referring to a noun in the context sentence, split by the noun gender and number. Higher gains can be observed for feminine a plural nouns. Taken from [69]

.

Evaluation of the proposed model was performed in terms of BLEU score, where a moderate improvement (0.7 BLEU) was observed. Also, evaluation using a random context sentence instead of the proper one was performed, to see whether the BLEU gains are due to correct context utilization, or some other effects have played role. A more fine-grained analysis of the system was also carried out. By investigating the distribution of attention over the context and source sentences, the authors discovered that a large part of the attention is given to the context sentence when pronouns like *„it“, „yours“, „ones“ and „I“* are being processed. These pronouns are indeed often hard to disambiguate without the additional context when translating for English to Russian (similarly so from English to Czech).

To see if this higher attention translates into higher BLEU scores, the authors created a pronoun disambiguation test set. Standford CoreNLP [40] was used to find sentence pairs containing coreferential pronouns, as such sentences are more likely to benefit from a context-aware translation. The sentences were extracted from held out part of the training corpus, OpenSubtitles2018. An issue observed with this approach was that most of the antecedents of the pronouns were also pronouns, which probably do not provide the information needed for the disambiguation. However, even if such sentences were left in the dataset, a bigger BLEU gain than on the original test set was seen. When considering only sentences with pronouns that have a noun as their antecedent in the previous sentence, even larger improvements can be seen.

The most interesting case is the pronoun *„it“* (Table 2.1). For further analysis of sentences containing this pronoun, Berkley word aligner was used to divide the test sentence pairs that contain *„it“* referring to a noun into parts based on the gender and the number of the noun. Larger improvements can be seen when *„it“* is to be translated into feminine or plural form, since in the training data, *„it“* is mostly masculine and the models tends to translate the pronoun into its masculine form when no context information is available. This observation links the context-aware NMT research with another currently actively researched NMT topic – gender bias in NMT models [67].

The results on pronoun translation suggest that the model learns to correctly determine an antecedent of the pronoun, i.e. perform anaphora resolution. The authors analyzed the attention weights (average of attention to context words over all attention heads) to asses this assumption. Again, only sentence pairs where a pronoun in the source sentence has a noun antecedent in the context sentence were used. Next, the cases where the context sentence only contained one noun, were excluded. The context word which had the highest attention weight was compared to an antecedent determined by CoreNLP coreference resolution system. Also, picking first, last, or random noun was evaluated as a simple heuristic.

Pronoun	Random	First	Last	Attention
it	40	36	52	**58**
you	42	63	29	**67**
I	39	56	35	**62**

Table 2.2: Agreement (in %) of a noun picked from the context sentence by attention or one of the three heuristics, with a noun determined by CoreNLP to be the pronoun antecedent. Attention has the largest agreement, suggesting that the model learns to perform anaphora resolution. Taken from [69].

Method	Agreement(%)
CoreNLP	77
attention	72
last	54

Table 2.3: Agreement (in %) of a noun picked from the context sentence as an antecedent by attention, by CoreNLP or the last noun, with a human judgement. Taken from [69].

The results are presented in Table 2.2. They show that the attention does indeed agree with CoreNLP in more cases than any of the heuristics. The authors also performed human-based evaluation, with results shown in Table 2.3. The context attention overall performs slightly worse than the CoreNLP system, however, the results show that the model learns to perform anaphora resolution.

In a work by **Agrawal et al.** [1], authors evaluate RNN and Transformer architectures with context windows of up to three previous source sentences and a next source sentence on the source side, and previous one or two target sentences on the target side. Context sentences were added either by concatenation (separated by a special token), or as an input for additional encoder. Models were trained and evaluated on English-Italian IWSLT 2017 dataset, consisting of transcribed TED talks.

A drop in BLEU score was observed when adding context to RNN via simple concatenation, probably because even though LSTMs have gating mechanisms and the network used attention, signal is still vanishing in long-range dependencies. When using multi-encoder architecture, BLEU increased for RNN. Other research suggests gains for RNNs even when using concatenation, however on OpenSubtitles dataset, where average sentence length is much shorter. For the Transformer, where only concatenation experiments were carried out, the best combination was one previous and one following source sentence on the source side and one previous target sentence on the target side, yielding a 2 BLEU gain over the baseline.

A work by **Maruf and Haffari** [41] is different from the others since the authors consider global context. Memory networks are used to store and use the context information. A recent follow up work [42] is using the Transformer and a sparse hierarchical attention to encode document-level context.

Recently, **Jean and Cho** [20] discovered that many of the models presented in previous work do not rely on the context information too much when translating a source sentence. The authors found this through a simple experiment – they replaced the context sentence with a random sentence from the dataset. A surprisingly low drop in BLEU score was observed.

A novel, multi level pair-wise loss function is presented. This loss function adds penalty to the training loss when the usual cross-entropy loss for a sentence with correct context is not significantly better (or even worse) than the loss for the same sentence with a randomly sampled context. In another words, the model is encouraged to utilize the context by the fact that a correct translation paired with a correct context is assigned lower error than a correct translation paired with a random context. Traditionally, NMT models that incorporate context are trained to maximize the log-likelihood on the training dataset $\mathcal{D}^{(tr)} = \mathcal{X}^{(tr)} * \mathcal{Y}^{(tr)} * \mathcal{C}^{(tr)} = \{(X_1, Y_1, C_1), \ldots, (X_N, Y_N, C_N)\}$., where X are the source sentences, C are the corresponding context sentences and Y are the reference target sentence. Log-likelihood is then defined as:

$$\mathcal{L}(\theta; \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \log p\left(y_t^n | y_{<t}^n, X_N, C_N\right) \tag{2.6}$$

where θ is the set of the model parameters.

Using the law of total probability, given a source sentence X, the additional context influence is neutral over the entire distribution of C:

$$p_\theta\left(y_t | y_{<t}, X\right) = \sum_C p_\theta\left(y_t | y_{<t}, X, C\right) p(C|X) \quad = \mathbb{E}_{C \sim C|X}\left[p_\theta\left(y_t | y_{<t}, X, C\right)\right] \tag{2.7}$$

When an additional context is used, there are two possibilities – either the context is useful or harmful. For useful context, the model is able to assign better probability to a correct translation (target token y_t^*) in case the context is provided than in case it is not.

Harmful context refers to the inverse situation – correct translation is assigned higher probability when such context is not provided. Equations above are defined at the token level, but can be extended to a sentence a dataset level functions easily:

$$\begin{gathered} s^{\text{tok}}\left(y_t|\cdot\right) = \log p_\theta\left(y_t^*|\cdot\right) \\ s^{\text{sent}}(Y|\cdot) = \textstyle\sum_{t=1}^{T} \log p_\theta\left(y_t^* | y_{<t}, \cdot\right) \\ s^{\text{data}}(\mathcal{Y}|\cdot) = \textstyle\sum_{Y \in \mathcal{Y}} s^{\text{sent}}(Y|\cdot) \end{gathered} \tag{2.8}$$

Authors propose to regularize training with these equations for all three levels, based on margin-ranking loss [10]:

$$\begin{aligned} \mathcal{R}(\theta; \mathcal{D}) = & \alpha_d \left[\left(\textstyle\sum_{n=1}^{N} T_n\right) \delta_d - s^{\text{data}}(\mathcal{Y}|\mathcal{X}, \mathcal{C}) + s^{\text{data}}(\mathcal{Y}|\mathcal{X})\right]_+ \\ & + \alpha_s \textstyle\sum_{n=1}^{N} \left[T_n \delta_s - s^{\text{sent}}\left(Y_n | X_n, C_n\right) + s^{\text{sent}}\left(Y_n | X_n\right)\right]_+ \\ & + \alpha_\tau \textstyle\sum_{n=1}^{N} \sum_{t=1}^{T_n} \left[\delta_\tau - s^{\text{tok}}\left(y_t^n | y_{<t}^n, X_n, C_n\right) + s^{\text{tok}}\left(y_t^n | y_{<t}^n, X_n\right)\right]_+ \end{aligned} \tag{2.9}$$

where the hyperparameters δ_d, δ_s and δ_t are margins and α_d, α_s and α_t are regularization strengths for data, sentence a token level respectively.

One of the issues is estimating the score that would be given to the translation without the additional context, since this would require to compute the score over all possible context sentences and the knowledge of $p(C|X)$. Therefore, a simple approximation is used, based on a assumption that X and C are independently distributed $p(C|X) = p(C)$ and that the distribution of C follows the distribution of dataset D. Then, the context-less probability can be estimated as:

$$s(\cdot|\cdot) = \log p(\cdot\cdot) \approx \log \frac{1}{M} \sum_{m=1}^{M} p\left(\cdot|\cdot, C_m\right) \tag{2.10}$$

In the experiments, the authors set M=1, i.e. the shuffled the currently processed minibatch and chose one random context sentence for each source sentence. The definitions of

Model	Correct context	Random context	$\Delta_{BLEU}^{\mathcal{D}^{\text{test}}}(\theta)$
Noncontextual baseline	29.16	-	-
Context-aware transformer	29.34	28.94	0.4
+ Context-aware learning	**29.91**	26.17	**3.74**

Table 2.4: Results (BLEU scores) of context-aware loss regularization applied to context-aware Transformer [69]. With the proposed regularization, context-aware Transformer outperforms the non-regularized model in terms of BLEU, and a higher drop in BLEU score can be observed when random context is supplied. This suggests that the regularized model depends more on the context sentence and it is able to utilize context information more effectively. Taken from [20].

useful and neutral context described above can also serve as an evaluation metric: During the training with the proposed algorithm, difference of scores for correct and random contexts can be computed on a validation set to see the progress of training. Similarily, another metric can be defined, this time using BLEU scores – difference of BLEU scores of translations using the correct and using a random context. Using these metrics can help us understand how much does the model depend on the context information.

Experiments (see Table 2.4) were carried out using a model proposed by Voita et al. [69], which was already described earlier. Based on tuning on validation set, the hyperparameters for the proposed loss were set to $\alpha_\tau = \alpha_d = 1, \alpha_s = 0, \delta_\tau = \delta_s = 0, \delta_d = \log(1.1)$. They observed a negligible gain in BLEU score on the validation set and no gain on the test set using a context-aware model trained only to maximize log-likelihood without the proposed modifications. Also, when random context is supplied to this model, only a small (0.4) BLEU drop can be observed.

When the proposed regularization is used, the model outperforms the baseline and a big drop in BLEU score is seen when random context is supplied instead of the correct one. This suggests that the model relies more on the context sentence and it is able to use the context to improve overall translation quality. Furthermore, as the modification affects purely the learning objective, this algorithm can be used in combination with virtually any context-aware NMT model.

Chapter 3

Training and evaluation datasets

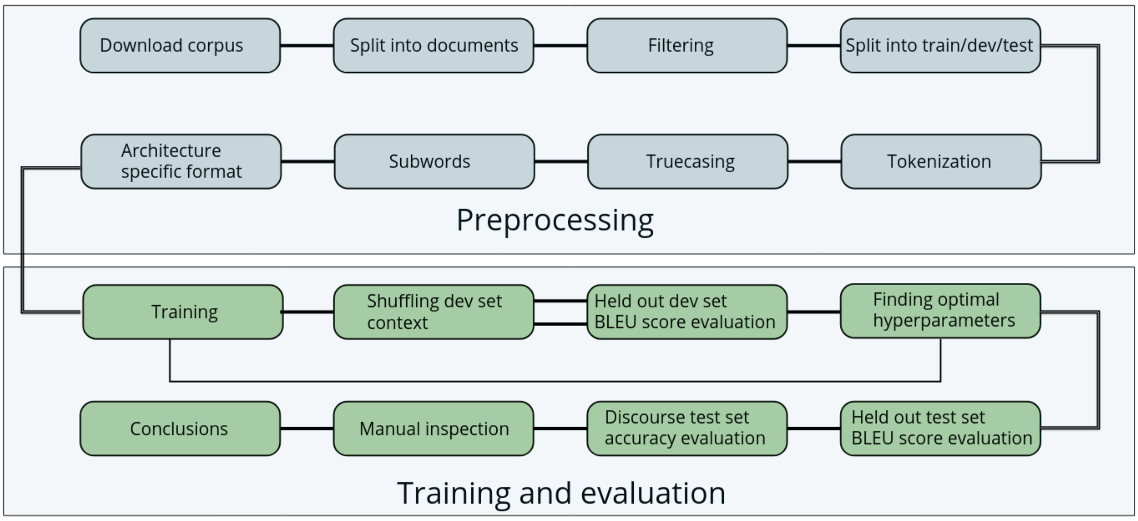

Download corpus
Split into documents
Filtering
Split into train/dev/test
Architecture specific format
Subwords
Truecasing
Tokenization
Preprocessing
Training
Shuffling dev set context
Held out dev set BLEU score evaluation
Finding optimal hyperparameters
Conclusions
Manual inspection
Discourse test set accuracy evaluation
Held out test set BLEU score evaluation
Training and evaluation

Figure 3.1: Pipeline used in this work. Steps are explained in depth in current and the two following chapters.

To compare the effects of utilizing context in different domains and with different types of data, three publicly available datasets were be used in the experiments: Europarl[1], OpenSubtitles2018[2] and WMT19 datasets for the news translation task[3], including Paracrawl[4], which was preprocessed in a specific way, described later. The datasets were used to train models for English to Czech and English to French translation, WMT18 dataset was used for English to Czech only. Overview of the complete workflow is presented in Figure 3.1. Sizes of individual datasets are provided in Table 3.1.

3.1 Europarl

Europarl [31] is a parallel corpus containing 21 EU languages, extracted from translations of European Parliament proceedings from 1996 to 2011. Since it is a content created by

[1] http://opus.nlpl.eu/Europarl.php
[2] http://opus.nlpl.eu/OpenSubtitles2018.php
[3] http://www.statmt.org/wmt19/translation-task.html
[4] https://paracrawl.eu/

Dataset	# lines	# words	# doc
OpenSub train	41.8M	678M	55.6k
OpenSub dev	7318	148k	20
OpenSub test	15684	262k	20
Europarl train	2M	115M	3138
Europarl dev	42k	2.5M	20
Europarl test	38k	2.3M	20

Dataset	# lines	# words	# doc
OpenSub train	42.4M	636M	56.4k
OpenSub dev	13.4k	214k	20
OpenSub test	14.k	227k	20
Europarl train	625k	31.4M	2060
Europarl dev	7144	358k	20
Europarl test	5170	258k	20
WMT19	58M	1.2G	-
News CZ 07-18	73M	2.3G	-

Table 3.1: Approximate sizes of datasets used in this work.

professional translators and the sentences are generally aligned as they should be, the filtering step is omitted during the preprocessing. In this work, English to French and English to Czech parts of the corpus are used.

3.1.1 Download

The corpus was downloaded from OPUS[5]. Five files are needed - three archives containing the documents in Czech, English and French, and two XML files with alignment info for the two language pairs. A toolkit called *uplug*, that serves for preprocessing and format conversion, is supplied alongside the OPUS corpora. In this work, script *uplug-readalign* is used to extract parallel documents using the XML alignment file.

3.1.2 Splitting into documents

Uplug-readalign generates a long parallel file with all the parallel sentences for one language pair, while preserving the document boundaries. For the next steps, two simple Python scripts were created - *readalign_to_docs.py* changes the format of input file to slightly more convenient one, and also splits the documents into training, validation and test sets. Second one, *docs2context.py*, adds desired context (specified by command line parameters) to the sentences and generates files in a format which is suitable for direct usage as a training and test data for Marian (after applying BPE, see the next section).

3.1.3 Byte pair encoding

An NMT system encoder converts source words into indices in the source vocabulary and subsequently into embeddings, also called word vectors. In the last layer of the decoder, probability distribution computed by a softmax function over the outputs of the layer, is used to choose a word from target vocabulary. There are two issues with this approach, especially for languages that use agglutination or compounding to form a new words, or are otherwise morphologically rich.

First, it would be prohibitively expensive computationally- and memory-wise to use a huge vocabulary that would cover most of the words used in the language – reasonable sizes are in order of tens of thousands of words. Second, even it would be possible to use such a big vocabulary, most of the words would appear very sparsely in the training corpora.

[5] http://opus.nlpl.eu/Europarl.php

With only few examples for a word, it is difficult for the network to learn a good quality embedding. For example, words like *fotbal* and *fotbalový* (meaning football as a noun, and football as an adjective) would not share any part of their representation, the word vectors would be learned completely independently during the training.

Until character-level (or even better, byte-level) NMT systems are fast enough to be used in production, it is necessary to mitigate this problem. The solution is to split the words into smaller parts, called subwords or word pieces, which would reduce both of the issues, e.g. *fotbal* -> *fot bal*, *fotbalový* -> *fot bal ový*. The technique used in this work is called byte pair encoding (BPE), originally presented as a simple compression algorithm by Philip Gage [14].

BPE was adapted for use in NLP by Sennrich et al. [58]. First, a vocabulary of all characters in the training corpus is created and the sentences in the training data are split into characters. Then all symbol pairs are iteratively counted and the most frequent one is merged into one symbol, e.g. if *i* is frequently followed by *n*, *in* is added to the vocabulary. In the next iteration, if *in* is often followed by *g*, *ing* is added to the vocabulary. Number of these merge operations is a hyperparameter of the algorithm. The outcome of this algorithm is that a most frequent words are kept whole, while rare words are split into more common, smaller units. For all of the experiments, byte pair encoding is applied using subword-nmt[6] with 30000 merge operations.

3.1.4 Sentence length analysis

Since in some of the experiments, sentences are concatenated and thus the input sequence length is multiplied, it is useful to know distribution of sentence lengths, so the shortest possible maximum length can be used, without loosing too many training sentences. Python script `hist_length.py` generates a histogram of source sentence lengths in subwords and prints out percentage of sentences longer than a number of token specified by a parameter. For Europarl, maximum length of 80 subword units was chosen based on this analysis, which led to leaving out leaving out 1.2 % of the training sentences for both English-Czech and English-French language pairs. As discussed later, this choice was wrong and had to be adjusted for a fair comparison of the models.

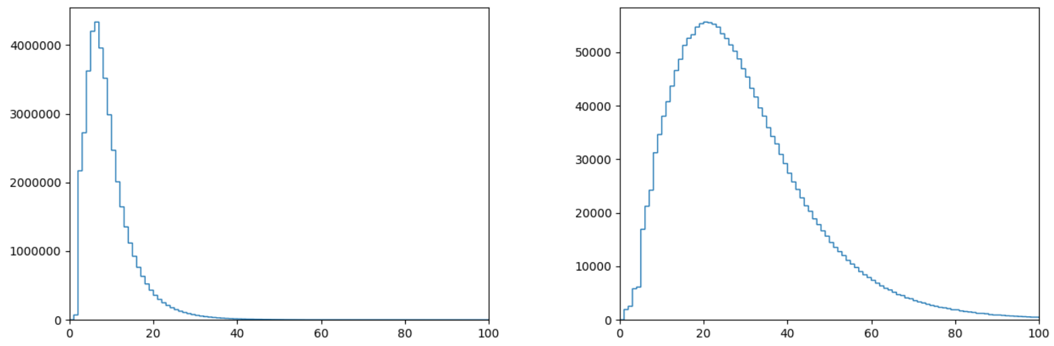
3.2 OpenSubtitles

Preprocessing of OpenSubtitles generally follows the same steps specified above for Europarl, with the difference of maximum sentence length allowed - as sentences are shorter in subtitles, 55 subword units is used as a limit, leaving out 0.02% of training examples for English-French and 0.03% for English-Czech.

3.3 WMT

It is very common in NMT research to see a new method that performs well on a small dataset, however, when evaluated on larger corpus, the gains vanish. To see if potential improvements on smaller datasets scale on larger data, a corpus used for training state-of-the-art NMT models was used. Conference on Machine Translation (WMT, based on the previous name „Workshop on Machine Translation“) is an annual conference accompanied

[6] github.com/subword-nmt



(a) OpenSubtiles

(b) Europarl

Figure 3.2: Source sentence lengths in subwords for English-French Europarl and OpenSubtiltes.

by an evaluation campaign. Training data for many tasks (e.g. translation in different domains, quality estimation, automatic post editing) and many language pairs are provided. Research groups can submit their systems, and results are evaluated by the participating researchers.

In this work, models were trained on data provided for English-Czech translation of news. The parallel data consist of Europarl, CzEng[7], Common crawl, News commentary and Paracrawl datasets. Monolingual datasets are also provided, Czech News crawl 2007-2018 was used in this work. To make use of the target language monolingual data, backtranslation [57] was performed, i.e. the monolingual News Commentary corpus was translated into English by an NMT model to create a parallel dataset.

The Paracrawl dataset provided by the organizers is filtered and does not preserve document boundaries, which is making this version unsuitable for training a document-level system. For this reason, the raw corpus was downloaded a preprocessed by a specific pipeline, described in the next section.

3.4 Paracrawl

Paracrawl is a recent project which is co-financed by the EU. It is a collection of parallel corpora, created by web crawling, and once finished, it will contain corpora for pairs of all official EU languages (as well as some others) and English. Since the Paracrawl corpus is composed with emphasis on recall, most of its contents is noise. For this reason, the preprocessing is more involved than for the other two corpora. The advantage of this corpus is that each sentence pair has a source and target URL address, so the corpus can be split into documents. A fairly new technique for scoring translation adequacy is used to score sentences and compute average score for the whole documents.

Documents with a low average score are filtered out, as well as documents that are deemed too different from a news domain (like car salesmen websites with lot of listings of offered cars, the translations can be adequate, but not really fit to be NMT training data),

[7] http://ufal.mff.cuni.cz/czeng/czeng17

documents recognized as a different language than the expected one, or documents that do not pass through few other simple heuristics.

3.4.1 Download

Czech Paracrawl version 1 (unfiltered) and monolingual News Crawl[8](for training a language model later on, news2014 to news2018) corpora are used in this part. The raw Paracrawl comprises of roughly 820 million sentences for English to Czech.

3.4.2 Deduplication

First step is to deduplicate the sentence pairs, since it would be wasteful to run computationally expensive language identification and adequacy scoring more times on the same sentence pairs. This is done using a simple awk command.

3.4.3 Language filter

Due to high noise ratio, it is essential to carry out language identification and filtering. In this work, langid.py [39] is used to perform this task. This script implements a naive Bayes classifier, using 1 to 4-grams of bytes as an input. Pre-trained models for nearly one hundred languages are available, including English, French and Czech. Using langind, language of each sentence is estimated and sentences with other than expected language are filtered out. For Czech, sentences classified as Slovak or Slovene are also admitted to improve recall.

3.4.4 Scoring

For each sentence pair in deduplicated and filtered data, adequacy and domain score is estimated by technique presented in a paper by Junczys-Dowmunt [23] – dual conditional cross-entropy filtering.

Dual conditional cross-entropy filtering is a novel method of filtering noisy parallel data, based on difference and absolute value of cross-entropy scores of the sentence pairs scored by two inverse NMT models. It is often crucial to apply some kind of filtering based on sentence pair adequacy, as NMT is very sensitive to data noise [29] [6]. A similar concept was used before for selecting in-domain monolingual data to train language models. This method is very straightforward – first it is necessary to train models for the desired language pair on „clean“ (WMT18 is this case) data in both directions. The next step is to use the models to score sentence pairs from noisy corpus (e.g. Paracrawl) with word-normalized cross-entropy. The score is computed as

$$H = |A - B| + \frac{|A + B|}{2}$$

where A and B are the cross-entropy scores. The first part of the formula aims to maximize agreement on how probable is that one side is translation of the other. The second part penalizes sentences where both sides are equally probable, but the probability is very low. This formula produces positive values, with zero being the best one. To obtain final

[8] http://www.statmt.org/wmt18/translation-task.html

adequacy score within 0-1 range, where 1 is the most adequate translation, the result is negated and exponentiated:

$$adq = exp\left(-\left[|A - B| + \frac{|A + B|}{2}\right]\right)$$

Using this score, highly adequate sentence pairs can be obtained from the training corpus. However, large percentage of adequate sentences is not fit as NMT training data, e.g. product names from e-shops, headers and footers, dates, menu items and so on. To filter out this part of data, very similar concept as above can be used. First, it is necessary to train two language models for the target language - one on all of the data we want to apply the filtering to, another one on clean data similar to filtered sentences we are expecting. In this work, WMT monolingual News corpus will be used as an approximation of general domain clean data. After training the models on Paracrawl and News, *domain score* is computed as follows:

$$dom = exp(-[A - B])$$

$$dom = max(dom, 1)$$

Where A a B are the cross-entropy scores of general (Paracrawl) and in-domain (News) LM for given target sentence. The values are clipped at 1 for further processing to prevent high domain score outweigh low adequacy score, as both of them are multiplied to obtain the final score:

$$score = dom \times adq$$

This method was used in WMT18 En-De translation winning system[24] and also won WMT18 data filtering shared task [23].

Since the unfiltered corpus is very large and contains a lot of duplicate sentence pairs, scoring the whole corpus would be very time consuming and wasteful. Thus, only a subset of the sentences, obtained by deduplicating and language filtering, was scored. However, for context-aware models training, whole documents, without any missing sentences are needed. The scored sentences from filtered and unfiltered corpora need to be matched, and the scores need to be mapped back into the unfiltered version.

For this purpose, first a hash of concatenated source and target sentence from the filtered, scored corpus is made, using `hash.py` script. A hash table that assigns a score to each hash is created. Then the whole unfiltered corpus is iterated over, each sentence pair is hashed and looked up in the hashtable `(match_hashes.py)`. If found, the score is added to the sentence pair. If not found, meaning that this sentence pair was removed by the language filter, the score is set to zero.

After having the complete, document-segmented corpus scored, average scores for each documents are computed. Documents with a score above a certain threshold are then selected and preprocessed in the same manner as Europarl and OpenSubtitles corpora described above.

Chapter 4

Proposed models

This chapter describes the context-aware NMT architectures which where evaluated in this work. Their possible advantages as well as shortcomings are discussed.

4.1 Baselines

The Transformer [68] and RNN network with GRU cells were used as a baseline. The base Transformer model was already described earlier, in Chapter 2. For Transformer, the hyperparameters were generally the same as in the original paper. A baseline Transformer architecture is presented in Figure 4.1. For the RNN model, the systems were similar to WMT2017 systems by University of Edinburgh [54]. Training scripts with the exact parameters can be found on the attached CD, or at `https://github.com/cepin19/dp`.

4.2 Concatenation

The most straight-forward approach to employ extended context is to simply concatenate multiple sentences as an input of the model. Main advantage of this approach is the simplicity – the changes are made solely on the preprocessing level. To inform the model which part of the input is supposed to be translated, and which part should be treated only as an additional context, the sentences can be separated by a special token. Another way of differentiating between source and context sentences is to use binary flag for each input token (also called *features* or *factors* in the literature [15] [56]), as in a paper by Voita et al. [69]. This method was not evaluated in this work, since the used framework lacks implementation of such input factors.

First possible downside to using concatenation is the increased sequence length – resulting in higher memory usage, and longer computation time for RNN models, since the input is processed sequentially, token by token, by these models. Additionally, RNNs suffer from problems with long-term dependencies, even if gating mechanisms (LSTM, GRU), and attention, which should mitigate them, are employed. The reason behind this is is in inherently sequential nature of RNNs. The number of steps between processing two tokens is linear in regard to their distance in the input sequence, and in each step, part of the information about the first token vanishes. For the Transformer, the sequence length does not pose such problem, since there is no notion of distance between input symbols (position representation of each token is added explicitly), all the input symbols are processed simul-

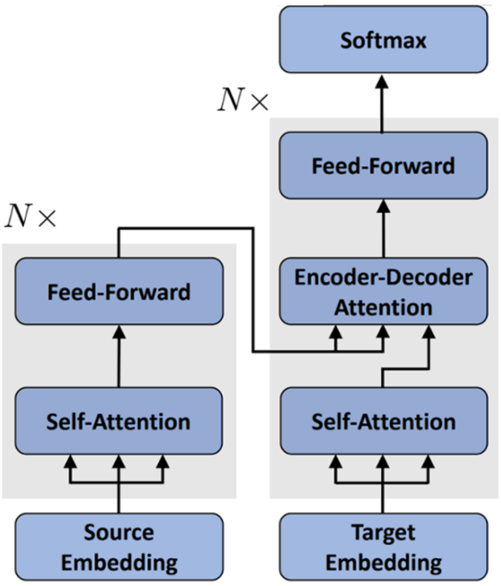


Figure 4.1: Schematic overview of the vanilla Transfomer model, used as a baseline and for the concatenation experiments. Taken from [73].

taneously and are connected directly by the network. The experimental results presented later support this hypothesis.

Another possible downside is that a large number of the available training corpora either do not contain document boundaries, or, even worse, are shuffled. Training examples from such corpora have no context information. It is possible to specify a „null“ context for these sentences and either mix them with sentences from document split corpora, or first train on these sentences and then fine tune the model on the sentences from document split corpora. However, experiences from domain adaptation and similar tasks show that NMT models are prone to the *catastrophic forgetting* problem [59] [3] [48] [71], possibly leading to suboptimal training data utilization when using this approach.

Finally, also due to the same forgetting problem mentioned above, it is not easily possible to use already trained sentence-level models, context-aware models using concatenation have to be trained from scratch. This can be a costly procedure in scenarios with many language pairs or domains.

4.3 Multiple encoders

Another way to integrate the context into an NMT model is via an additional encoder, which may or may not have a same structure as the original one.

4.3.1 Two identical encoders

The simplest multiple encoder model employs two encoders with identical structure. Source sentence is fed into the original encoder, and context sentence into the additional one. En-

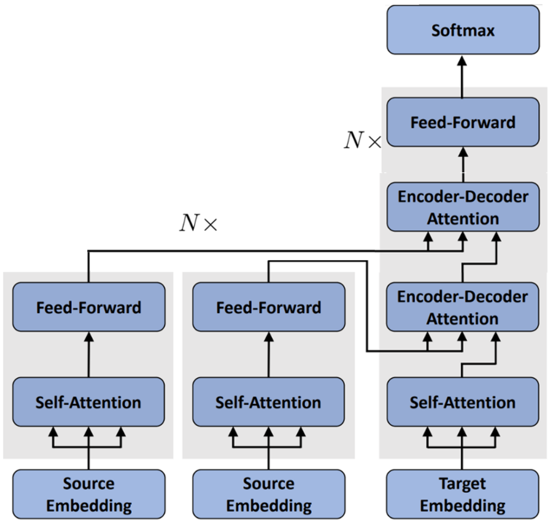


Figure 4.2: Schematic overview of the dual encoder Transformer model with serial encoders-decoder attention.

coding runs independently for both of the encoders. Encoders can have either separate, or shared layers, i.e. weights of the encoder neurons are the same in both encoders, identical weight matrices are used. For dual encoder with shared layers, a special token may be concatenated to the context sentence, so that the encoder learns to treat context sentences differently than source sentences. In decoder, the only difference in regard to the vanilla model is that source-target attention the decoder attends over both encoders. Let $c_i^{(1)}$ and $c_i^{(2)}$ be the context vectors produced by an attention over source and context encoder hidden states, respectively. The decoder needs to by able to combine attention information from both encoders, and there are a several attention strategies for multiple encoders. Possible combination methods for RNNs are discussed for example in [5]. The first presented approach is to concatenate the context vectors created by the encoders, use linear transformation to resize the vector back to its original size and compute the attention over this vector:

$$c_i = W_c \left[c_i^{(1)}; c_i^{(2)} \right] + b_c$$

W_c and b_c are the learned weights and biases, and c_i denotes the final context vector used by the decoder. Another way is to use for example hierarchical attention, introduced by Libovický and Helcl [37], which performed well in [5]. Other possibilities of multi-encoder attention in RNNs are not discussed here, since my work is mainly centered around the Transformer model.

For the Transformer, Libovický and Helcl [38] propose three multiple encoder attention strategies. The first one, which is called *serial* in the paper, is presented in Figure 4.2. In this approach, the attention sub-layer is simply repeated for each encoder. In another words, an encoder-decoder attention is first computed for the first (source) encoder. Let $\mathbf{C^1}$ and $\mathbf{C^2}$ be states of the first (source) and second (context) encoders respectively, and $\mathbf{S^{i-1}}$ output of the previous layer of the decoder. This output vector, already containing information about which parts of the source sentences are attended to in the current step of decoding, is used as a *query* for the encoder-decoder attention over the second encoder

($\mathbf{D^i}$). One layer of the decoder the performs these operations:

$$\begin{aligned}
\mathbf{A}^i_{self} &= MultiHeadAtt\left(\boldsymbol{S}^{i-1}, \boldsymbol{S}^{i-1}, \boldsymbol{S}^{i-1}\right)\\
\mathbf{A}^i_{self} &= LayerNorm(\boldsymbol{S}^{i-1} + \mathbf{A}^i_{self})\\
\mathbf{A}^i_{enc1} &= MultiHeadAtt\left(\boldsymbol{A}^i_{self}, \boldsymbol{C}^1, \boldsymbol{C}^1\right)\\
\mathbf{A}^i_{enc1} &= LayerNorm(\boldsymbol{A}^i_{self} + \mathbf{A}^i_{enc1})\\
\mathbf{A}^i_{enc2} &= MultiHeadAtt\left(\boldsymbol{A}^i_{enc1}, \boldsymbol{C}^2, \boldsymbol{C}^2\right)\\
\mathbf{A}^i_{enc2} &= LayerNorm(\boldsymbol{A}^i_{enc1} + \mathbf{A}^i_{enc2})\\
\mathbf{S}^i &= \left[\mathrm{FNN}\left(\mathbf{A}^i_{enc2[.,1]}\right); \ldots ; \mathrm{FNN}\left(\mathbf{A}^i_{enc2[\cdot,M]}\right)\right]\\
\mathbf{S}^i &= LayerNorm(\mathbf{A}^i_{enc2} + \boldsymbol{S}^i)
\end{aligned} \tag{4.1}$$

$\mathbf{A}^i_{self}$ is the result of decoder self-attention, $\mathbf{A}^i_{enc1}$ is the result of attention over the first encoder, $\mathbf{A}^i_{enc2}$ is the result of attention over the second encoder and $\mathbf{S}^i$ is the new decoder hidden state.

Another approaches presented in the work are *parallel*, *flat* and *hierarchical* attention. Since the authors did not observe significant differences between the performance of these strategies, my work only explores serial attention, i.e. decoder first attends over one encoder and then, with state already updated by this attention, attends over the second one.

4.3.2 Context encoder

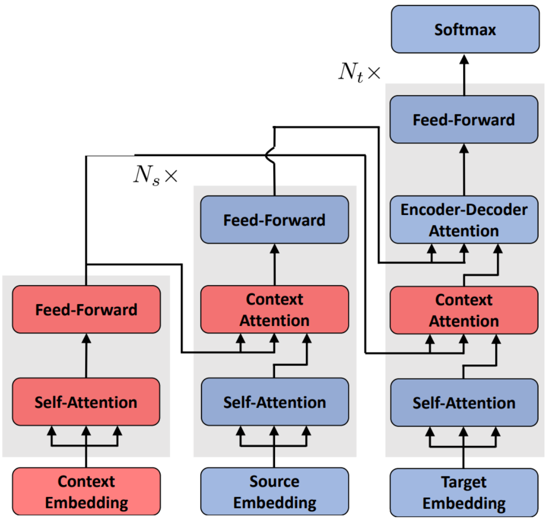


Figure 4.3: Schematic overview of the Transformer model with context encoder. Taken from [73].

Inspired by [73], I implemented Transformer with context encoder in Marian [25]. This architecture also utilizes two encoders, yet there are a few differences in comparison with multiple encoder architecture described above. First, the encoders are not exactly the same – the context encoder has fewer self-attention layers (only one, while the source encoder has six). Second, the context encoder states are also attended over in the source encoder, and not only in decoder, in contrast to the previous approach. Also, the influence of context encoder is gated by a sigmoid gate. This should allow better usage of the context. Schematic overview is presented in Figure 4.3.

The context encoder consists of N_c layers that have the same structure as in the original Transformer model – self-attention, followed by position-wise fully-connected feed forward layer. Residual connection and a layer normalization are used after each of the sublayers. Computation steps performed in each layer are then:

$$\begin{aligned}
\mathbf{A}^i &= \mathit{MultiHeadAtt}\left(\boldsymbol{C}^{i-1}, \boldsymbol{C}^{i-1}, \boldsymbol{C}^{i-1}\right) \\
\mathbf{A}^i &= \mathit{LayerNorm}(\boldsymbol{C}^{i-1} + \mathbf{A}^i) \\
\mathbf{C}^i &= \left[\mathrm{FNN}\left(\mathbf{A}^i_{\cdot,1}\right); \ldots ; \mathrm{FNN}\left(\mathbf{A}^i_{\cdot,M}\right)\right] \\
\mathbf{C}^i &= \mathit{LayerNorm}(\mathbf{A}^i + \boldsymbol{C}^i)
\end{aligned} \tag{4.2}$$

where C^i the annotation vector in layer i. The annotation vector from the last layer of context encoder is incorporated into both source encoder and decoder.

In the source encoder, an additional context attention layer is used in between self-attention and feed forward layer. Since the source sentence is usually more important for a correct translation than the context sentence, and since residual connections after the context attention may allow the context representations to influence the source representations uncontrollably, it may be beneficial to restrict the context encoder influence. For this reason, a gating mechanism is used for the residual connections after the context attention layer. Source encoder the performs this function:

$$\begin{aligned}
\mathbf{A}^i &= \mathit{MultiHeadAtt}\left(\boldsymbol{S}^{i-1}, \boldsymbol{S}^{i-1}, \boldsymbol{S}^{i-1}\right) \\
\mathbf{A}^i &= \mathit{LayerNorm}(\boldsymbol{S}^{i-1} + \mathbf{A}^i) \\
\mathbf{D}^i &= \mathit{MultiHeadAtt}\left(\boldsymbol{A}^i, \boldsymbol{C}^{N_c}, \boldsymbol{C}^{N_c}\right) \\
\mathbf{D}^i &= \mathit{LayerNorm}(\boldsymbol{A}^i + \mathbf{D}^i) \\
\mathbf{S}^i &= \left[\mathrm{FNN}\left(\mathbf{D}^i_{\cdot,1}\right); \ldots ; \mathrm{FNN}\left(\mathbf{D}^i_{\cdot,M}\right)\right] \\
\mathbf{S}^i &= \mathit{LayerNorm}(\lambda \boldsymbol{S}^i + (1-\lambda)\boldsymbol{D}^i)
\end{aligned} \tag{4.3}$$

where λ is the sigmoid gate with learned weights – $\mathbf{W}_i$ and $\mathbf{H} + \mathbf{W}_s$ are trainable weights of the gate function:

$$\lambda = \sigma\left(\mathbf{W}_i\mathbf{H} + \mathbf{W}_s\,\mathrm{SubLayer}(\mathbf{H})\right)$$

In the decoder, the context annotation vector is incorporated in the same manner. Each decoder layer has four sublayers – self-attention, source encoder-decoder attention, context encoder-decoder attention and a feed forward layer.

This architecture allows for a vanilla Transformer model to be pretrained on general data without document boundaries, which are usually much larger than a document-split training data. Then, the weights of this base model are frozen and the additional components (highlighted in red in Figure 4.3) are added. Their weights are then tuned on a smaller corpus with document level information. During inference, the system can either use the full model in case that the input has context information, or only the pretrained part, for single sentence translation.

4.3.3 Context-aware Transformer

The second architecture implemented in my work is the context-aware Transformer, presented by Voita et al. [69]. Experimental results and observations obtained in the original paper are summarized in Chapter 2, section Related work. Overview of the encoder is shown in Figure 2.6. The **source encoder** consists of N layers, with $N-1$ layers identical as in the original Transformer encoder. The last layer is used to incorporate the encoded context representations. Aside from the source encoder self-attention, this layer also performs attention over the context encoder representations. The outputs of these two multi-head attention blocks are combined together using a gated sum:

$$g_i = \sigma\left(W_g\left[c_i^{(s-attn)}, c_i^{(c-attn)}\right] + b_g\right) \tag{4.4}$$

$$c_i = g_i \odot c_i^{(s-attn)} + (1-g_i) \odot c_i^{(c-attn)} \tag{4.5}$$

where $c_i^{(c-attn)}$ is the attention output for the context encoder, $c_i^{(s-attn)}$ is the attention output for the source encoder, W_g and b_g are the learned weights and bias for the gate and c_i is the final, gated sum. A position-wise, feed-forward layer is applied to this sum and the resulting tensor is used in the decoder.

Structure of the **context encoder** is the same as the structure of the original Transformer encoder. First $N-1$ layers shared their weights with the corresponding layers in the source encoder, only the last layer has its own set of weights. Since most of the layers are shared, to make it possible for the encoders to distinguish whether source or context sentence is being encoded, a special token, denoted $<bos>$, is inserted at beginning of the context sentences. Decoder is identical to the original Transformer decoder.

Chapter 5

Implementation

The NMT framework used in this work is Marian [25], which is fast and efficient, written in pure C++ with minimal dependencies. It provides differentiation engine based on dynamic computation graphs, unified GPU/CPU interface, and implements most of the current state-of-the-art architectures and techniques in NMT, including both of the network architectures that are used in this work – RNN and Transformer.

The basic unit of the computation graph is `Chainable`, usually referenced by its pointer of type `Expr` in the code. All the building elements of the computation graph, like weight matrices, inputs and operations, are of type `Chainable`. At least two methods are usually implemented in a `Chainable` which represents an operation: `forward`, which performs the implemented function on the inputs, and `backward`, which computes the gradient for the weights updates.

In this work, most modifications are done on a higher level of abstraction and consist of combining already existing building blocks into an encoder-decoder model. A simplified example of the interface of encoder-decoder model follows:

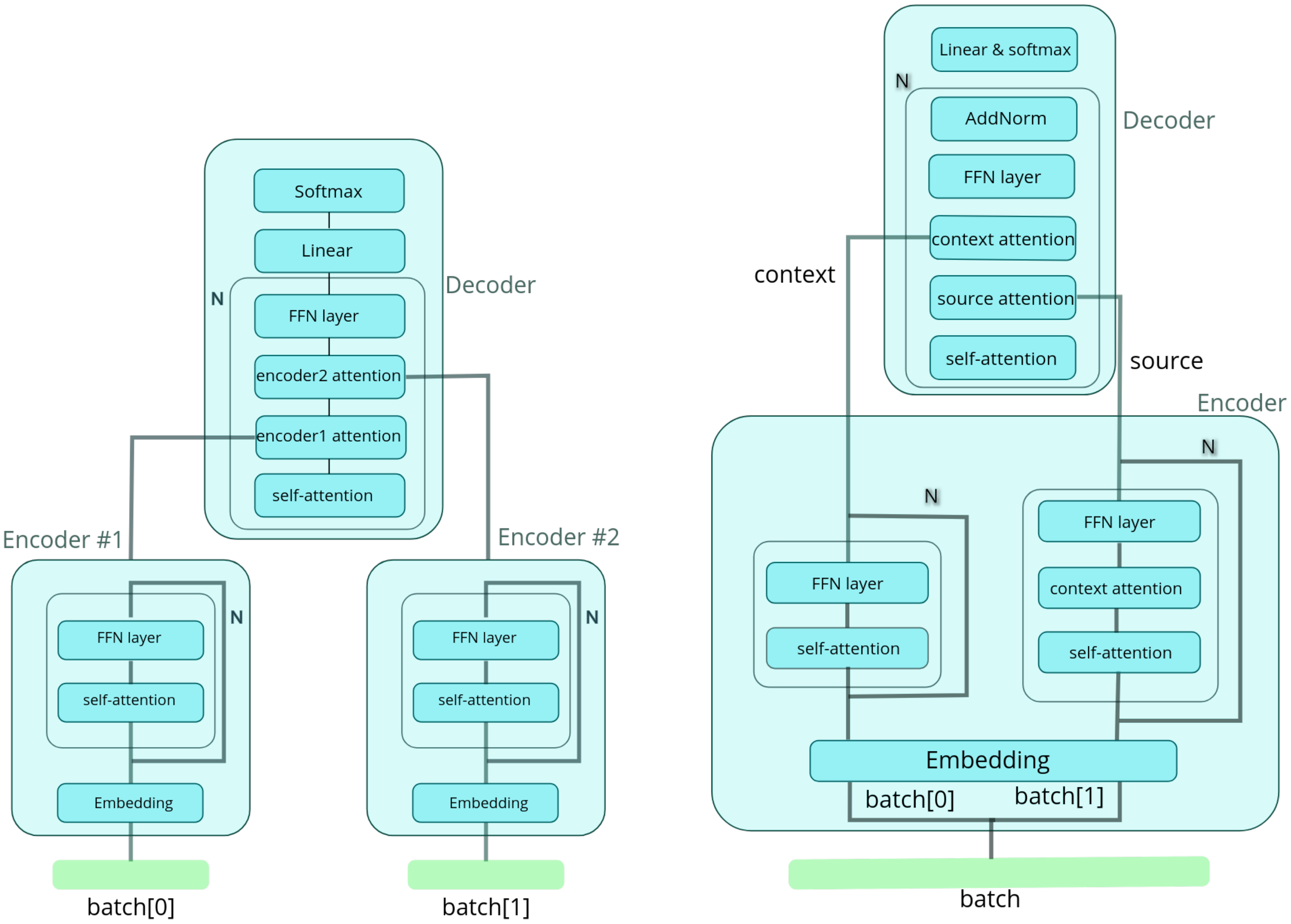
```
class Encoder {
    EncoderState build(batch);
};

class Decoder {
    DecoderState startState(EncoderState[], batch);
    DecoderState step(DecoderState);
}
```

Specific models like Transformer derive their encoder and decoder from a class similar to the one presented here, and implement the functionality. The `Encoder::build` creates the computation graph for the encoder and returns `EncoderState` based on the input inside the `batch`. Batch is a matrix containing vocabulary ids of words in the input data. `EncoderState` holds the context vector produced by the encoder and a mask for this vector.

The `Decoder::step` method uses `EncoderState` and `DecoderState` to generate logits (probabilities of all words in target vocabulary) for one step of the decoding, and then updates the `DecoderState`.

In the previous chapter, four approaches to employing context in NMT are proposed. First of them, *concatenation*, does not need any changes to the baseline model. The second



(a) Dual encoder

(b) Context encoder

Figure 5.1: Implementation of dual encoder and context encoder in Marian

one, *dual encoder* model with identical encoders, is already implemented in Marian, using serial encoders-decoder attention. Thus, only the *context encoder* by Zhang et al. [73] and *context-aware Transformer* by Voita et al. [69] were actually implemented.

5.1 Marian source structure

Marian only has an automatically generated, Doxygen code documentation. A bit outdated version available at `https://marian-nmt.github.io/docs/marian/classes.html`. A current version can be generated using `make doc` in the build directory.

Generally, two source files need to be edited to add a new model. First is the file containing the actual implementation, in case of the Transformer model it is located in `models/transformer.h`. All the models must be registered in `models/model_factory.cpp`. This registration links the constructor of the model with configuration options determining the model type – e.g. when the training parameter `type` is set to *transformer* in the config file or on the command line, the registration assures that the constructors for Transformer encoder and Transformer decoder are called. Source file `common/config_parser.h` also needed to be edited to add config options for freezing pretrained layers and controlling number of context encoder layers and gating. In my case, I also had to modify `model/states.h`, so that the `EncoderState` can also hold the context encoder context vector.

The original Transformer model is implemented in classes `EncoderTransformer` and `DecoderTransformer`, which are both derived from `Transformer` class with either `BaseEncoder` or `BaseDecoder` as a class template. The `Transformer` class implements the layers and functions utilized by a Transformer model, and only slight changes are made in these.

Most modifications are done in the `EncoderTransformer` class. Implementation of context encoder is described in the next section.

5.2 Context encoder

In Marian, multiple encoder models are supported, and it would seem intuitive to make use of their implementation. However, it is not easy to adapt the implementation to the needs of the context encoder model. The central issue is that the multiple encoders are meant to run in parallel, whereas in the context encoder architecture, a representation of the context sentences must be computed first, since it is used during encoding of the source sentence.

To get around this issue, I decided to create a new encoder class that implements both encoders, and returns two context vectors instead of one. Since this class is treated as a single encoder elsewhere in the code, I also had to adjust the decoder to attend over both context vectors generated by the encoders. Another issue that arises with this approach is how to divide the input between the encoders correctly, i.e. how to make the context encoder encode the context sentence and the source encoder encode the source sentence. Schematic illustration of the difference between implementing dual encoder as two separate encoders, and a context encoder as a single class, are presented in Figure 5.1. After resolving these issues, the implementation consists mostly of putting together already implemented building blocks (layers) and making small modification to them, to allow gating of the residual connections a freezing of the weights.

Two new classes were added to `transformer.h` – `EncoderTransformerContext` and `DecoderTransformerContext`, based on `EncoderTransformer` and `DecoderTransformer`. First, the issue with splitting the input between context encoder and source encoder needs to be solved. Input data are passed to the `apply` function through a parameter `batch`, which is of type `data::CorpusBatch`. It can be thought of as a vector which contains matrices of word ids for a batch from each of the input files, demonstrated in Table 5.1. In code, such batch would look as follows:

```
batch=[[[I0S1W1,I0S2W1,I0S3W1,I0S4W1],[I0S1W2,I0S2W2,I0S3W2,I0S4W2],...],\
[[I1S1W1,I1S2W1,I1S3W1,I1S4W1],[I1S1W2,I1S2W2,I1S3W2,I1S4W2],...]]
```

where I denotes the input file index, S is the sentence number, and W is the index of a word in the sentence. Hence, with values presented in the table, the contents of the batch vector would be:

```
batch=[[[53,13,457,235],[145,421,111,9888],...],\
[[31,10,7,15],[587,154,11,782],...]]
```

	W 1	W 2	W 3	W 4	W 5
S 1	53	145	1289	17	454
S 2	13	421	13	246	1134
S 3	457	111	22	586	5457
S 4	235	9888	553	988	4554

(a) batch[0] = one batch of sentences from the source file

	W 1	W 2	W 3	W 4	W 5
S 1	31	587	4588	1145	154
S 2	10	154	124	985	78
S 3	7	11	78	548	1
S 4	15	782	587	8	4

(b) batch[1] = one batch of sentences from the reference file

Table 5.1: Example of a batch structure, which holds input word ids, for two input files (source and target), batch size 4 and sentence length 5. Rows are sentences and columns are words in each sentence.

To select a batch of sentences belonging to one of the input files, the topmost index is used. For example, when using two encoders, there are three input files – two source files and one target file with reference translations. Then `batch[0]` contains batch of sentences from the first input file, `batch[1]` from the second input file and `batch[1]` contains the reference translations. Constructors of `EncoderTransformer` and `DecoderTransformer` can be passed an `index` parameter that is used to select the correct batch. However, I can't use this index, since both of the encoders are implemented in a single encoder class and are instantiated as a single object. To simplify the implementation, only two source files and one target file are considered – then it is safe to use fixed indices: 0 for context encoder, 1 for source encoder and 2 for decoder.

After having solved the input processing issues, the next step is to actually implement the network. All the building blocks (layers) all already implemented, hence it is only needed to put them together and make a few small modifications to enable gating and freezing of the weights.

First step of the encoding is to convert vocabulary word ids in the input batches into embeddings. `EncoderTransformer::createWordEmbeddingLayer()` method is used to create the embedding layer. The same embedding layer will be used word both encoders, since same vocabulary is used for all the inputs. After using `apply` method of the embedding layer on both batches, a tensor consisting of embeddings for all of the words is stored in variables `layer` for the source sentences and `layerContext` for the context sentences. From now on, there are two different paths in the computation graph.

For the **context encoder**, the usual Transformer encoding layer is applied n times, where n is defined by the `context-enc-depth` option, which had to be added in the `common/config_parser.cpp` file. The layer consists of two functions:

```
Expr Transformer::LayerAttention(std::string prefix, Expr query, const
Expr& keys, const Expr& values, const Expr& mask, bool cache = false,
bool saveAttentionWeights = false, bool trainable=true)

Expr Transformer::LayerFFN(std::string prefix, Expr input, std::string
op="", bool trainable=true)
```

The first function performs multi-head self-attention, residual connection after this operation, and also the final layer normalization. The `prefix` parameter is an identifier of the node in the computation graph. Tensors query, keys and values are inputs of the self-attention function as described in Chapter 2. In case of self-attention, all three input tensors are the same, and they are obtained either as output of the previous layer, or, in case of the first layer, output of embedding layer. The function returns tensor of the same dimensions as query and it is again stored into the layerContext variable.

This tensor is an input of the position-wise feed forward layer, represented by LayerFFN function. This layer applies the same linear transformation to each input position, i.e. to each input token representation stored in the input tensor. This transformation is followed by residual connection and layer normalization. Output of this sub-layer is again a tensor with the same dimensions, which is used as a query, key and value for the subsequent self-attention layer, or, in case of the last encoder layer, as a context vector used in the decoder. The output is stored into the layerContext variable. In a simplified code, the complete context encoder looks as follows:

```
Expr embedding=createWordEmbeddingLayer(0);
Expr layerContext=embedding->apply(0/*batch index*/);
auto contextEncDepth = opt<int>("context-enc-depth");
for (int i = 1; i <= contextEncDepth; ++i) {
    //multi head self-attention
    layerContext = LayerAttention("context_encoder_self_" + i,
                            layerContext, // query
                            layerContext, // keys
                            layerContext, // values
                            layerMaskContext);
    //position-wise feed forward network
    layerContext = LayerFFN("context_encoder_ffn" + i, layerContext);
}
```

The **source encoder** implementation is very similar, with a small number of modifications. Another sub-layer is placed between the self-attention and the position-wise feed forward layer. This sub-layer performs attention over the context representation created by the context encoder. This attention is implemented by the same function – LayerAttention, the only difference are the input parameters. Query tensor is obtained as the output of previous layer (or embedding layer in the first layer), which is stored in variable layer, keys and values are the context vector, output of the last layer of the context encoder, stored in layerContext.

Parameter trainable was added to the original function to allow freezing of some of the layers when training only the context part of the model. In the context encoder, this value is always set to true, since only weights for the pretrained original Transformer blocks are frozen during the context Transformer training. This parameter propagates deeper into the function creating nodes in the computation graph. On the lowest level, in file graph/nodes.h, trainable_ attribute of the node representing the weights is set to the bool value of the parameter.

The subsequent position-wise feed forward layer uses a gated residual connection, i.e. before summing the input and output tensors of the layer, they are weighted by a sigmoid gate. This is done to prevent uncontrolled influence of the context on the source encoder representations – usually, source sentence is far more important for the translation than the context sentence, and it might be beneficial to let the network learn to regulate the influence of the context encoder. In simplified code:

```
/*normal residual connection*/

//preprocessing of the input tensor, e.g. dropout
auto output = preProcess(prefix + "_Wo", opsPre, input, dropProb,
    trainable);

//position-wise FF layer
output = dense(output, prefix, /*suffix=*/std::to_string(depthFfn),
    dimModel,nullptr,0.0f,trainable);

/*residual connection*/
output = input+output;


/*gated residual connection*/
//sigmoid gate
Expr sigmoid_gate2(Expr x, Expr y, std::string prefix, std::string suffix,
    int outDim) {
    auto graph_x = x->graph();
    auto graph_y = y->graph();

    //trainable parameters
    auto Wi = graph_x->param(prefix + "_Wi" + suffix, {x->shape()[-1],
        outDim},inits::glorot_uniform);
    auto bi = graph_x->param(prefix + "_bi" + suffix, {1, outDim},
        inits::zeros);
    auto Ws = graph_y->param(prefix + "_Ws" + suffix, {x->shape()[-1],
        outDim}, inits::glorot_uniform);
    auto bs = graph_y->param(prefix + "_bs" + suffix, {1, outDim},
        inits::zeros);

    x = affine(x, Wi, bi);
    y = affine(y, Ws, bs);
    x = sigmoid(x + y);
    return x;
}
auto output = preProcess(prefix + "_Wo", opsPre, input, dropProb,
    trainable);
output = dense(output, prefix, /*suffix=*/std::to_string(depthFfn),
    dimModel,nullptr,0.0f,trainable);
auto lambda= sigmoid_gate(input,output,prefix,"lambda",dimModel);
//gated residual connection
output=lambda*output+(1-lambda)*input
  }
```

The desired training sequence is to first train the vanilla Transformer on sentence-level corpus, freeze the weights, add context-aware extensions and train only the newly added parameters on a document-level corpus. For this purpose, all of the layers of the source encoder, excluding the attention to the context encoder and the sigmoid gates, are frozen during the training on the document-level corpus.

After the encoder computation is completed, `EncoderState` is returned. `EncoderState` is implemented in `models/states.h` and holds the context vector produced by the encoder, corresponding mask, and pointer to the input batch, stored in private attributes `context_`, `mask_` and `batch_`. These values were stored in variables `layer`, `layerMask` and `batch` in the encoder `build()` method. The contents of the context vector and mask are accessible from the decoder by methods `getContext()` and `getMask()`. Since two encoders, each one with its own context vector and mask, are used, the EncoderState class was modified by adding new attributes to also hold the vectors produced by the context encoder:

```
/*original encoder state*/
EncoderState::EncoderState(Expr context, Expr mask, Ptr<data::CorpusBatch>
    batch);

EncoderTransformer::build(Ptr<data::CorpusBatch> batch){
    ... //computation
    return new<EncoderState>(layer,layerMask, batch);
}

/*modified encoder state for context encoder*/
EncoderState::EncoderState(Expr context, Expr mask, Expr documentContext,
    Expr documentMask, Ptr<data::CorpusBatch> batch);

EncoderTransformerContext::build(Ptr<data::CorpusBatch> batch){
    ... //computation
    return new<EncoderState>(layer,layerMask, layerContext,
        layerContextMask, batch);
}
```

The decoder was also slightly modified to attend to context vectors from both encoders. One more encoder-decoder attention layer was inserted after decoder self-attention. The original Transformer layers are frozen and residual connection after sub-layer following the context encoder attention sub-layer is gated:

```
/*original decoder*/
DecoderTransformer::step(<DecoderState> state){
    auto encoderContext = encoderState->getContext();
    auto encoderMask = encoderState->getMask();
    auto query = state->getTargetEmbeddings();
    for(int i = 0; i < decDepth; ++i) {
       //decoder self-attention, different function because future
          postions in decoder state need to be masked
       query = DecoderLayerSelfAttention(decoderState, prevDecoderState,
          name, query, selfMask, startPos);
       //encoder-decoder attention
        query = LayerAttention(name,
                             query,
                             encoderContext, // keys
                             encoderContext, // values
                             encoderMask);
        query = LayerFFN(prefix_ + "_l" + layerNo + "_ffn", query);
    }
    //probabilities of words in the target vocabulary for this step of
       decoding
    Expr logits = output_->apply(query);

    //update decoder state for the next step
    return New<TransformerState>(decoderStates, logits,
       state->getEncoderStates(), state->getBatch());

/*modified decoder*/
DecoderTransformerContext::step(<DecoderState> state){
    auto encoderContext = encoderState->getContext();
    auto encoderMask = encoderState->getMask();
    auto encoderDocumentContext = encoderState->getDocumentContext();
    auto encoderDocumentMask = encoderState->getDocumentMask();
    auto query = state->getTargetEmbeddings();
    for(int i = 0; i < decDepth; ++i) {
      //decoder self-attention, the weight are frozen
       query = DecoderLayerSelfAttention(decoderState, prevDecoderState,
          name, query, selfMask, startPos, !freeze);
       //context encoder-decoder attention
       query = LayerAttention(name,
                           query,
                             encoderDocumentContext, // keys
                             encoderDocumentContext, // values
                             encoderDocumentMask);
       // source encoder-decoder attention
      query = LayerAttentionGated(name,
                             query,
                             encoderContext, // keys
                             encoderContext, // values
                             encoderMask,!freeze);
        query = LayerFFN(prefix_ + "_l" + layerNo + "_ffn", query,
           !freeze);
    }
    //probabilities of words in the target vocabulary for this step of
       decoding
    Expr logits = output_->apply(query);
    //update decoder state for the next step
    return New<TransformerState>(decoderStates, logits,
        state->getEncoderStates(), state->getBatch());
}
```

5.3 Context-aware Transformer

The second model implemented in my work is the context-aware Transformer by Voita et al. [69]. The modifications are similar as above, with the difference that for this model, only the encoder had to be modified, while decoder stays the same.

The same approach as above was used for the input processing – encoder is implemented as a single class and the input batch is divided into source sentence and context in the same fashion as in the previous case. The structure of the encoder is different (see Figure 2.6) – there are no connections between the encoders up until the last layer, where states of the encoders are gated through a sigmoid gate and summed. The sigmoid gate was already implemented in the previous section. Only this single context vector is returned by the encoder, opposed to two context vectors, one for each encoder, returned by the previous model. The original Transformer decoder without any modifications can be used thanks to this fact.

The modified encoder is implemented by the `EncoderTransformerVoita` class. Simplified commented code follows:

```
/*compute word embeddings*/
Expr contextLayer=embedding->apply(0/*batch index*/);
Expr layer=embedding->apply(1);

//layer created in this loop are identical for both encoders and share the
    weights
for(int i = 1; i <= encDepth-1; ++i) {
    //context encoder self attention
    layerContext = LayerAttention(prefix_ + "_l" + std::to_string(i) +
        "_self",
                layerContext, // query
                layerContext, // keys
                layerContext, // values
                layerMaskContext);
     //source encoder self attention, weights are shared with the context
         encoder
     layer = LayerAttention(prefix_ + "_l" + std::to_string(i) + "_self",
            layer, // query
            layer, // keys
            layer, // values
            layerMask);
     layerContext = LayerFFN(prefix_ + "_l" + std::to_string(i) + "_ffn",
         layerContext);
     layer = LayerFFN(prefix_ + "_l" + std::to_string(i) + "_ffn", layer);
}
```

```
//final context encoder layer, not shared with the source encoder
layerContext = LayerAttention(prefix_ + "_l" +
    std::to_string(encDepth) + "context_self",
              layerContext, // query
              layerContext, // keys
              layerContext, // values
              layerMaskContext);
 layerContext = LayerFFN(prefix_ + "_l" + std::to_string(encDepth) +
    "context_ffn", layerContext);


 //attention over the context encoder with source encoder
    representations as queries
 auto layerContextSource = LayerAttention(prefix_ + "_l" +
    std::to_string(encDepth) + "context_src_self",
                        layer, // query
                        layerContext, // keys
                     layerContext, // values
                        layerMaskContext);

//final source encoder self-attention
 auto layerSource = LayerAttention(prefix_ + "_l" +
    std::to_string(encDepth+1) + "_self",
                   layer, // query
                   layer, // keys
                   layer, // values
                   layerMask);
 //gating function
 auto lambda=sigmoid_gate2(layerContextSource,layerSource,"sum_gate",\
 "lambda",layerContextSource->shape()[-1]);
 auto output=lambda*layerContextSource+(1-lambda)*layerSource;
 output = LayerFFN(prefix_ + "final_ffn", output);
 return New<EncoderState>(output, batchMask, batch)
```

5.4 Scripts

Aside from implementing the context-aware models, several support scripts were created or modified. They can be divided into two categories: preprocessing and evaluation.

5.4.1 Preprocessing

To split datasets into documents, filter them, and convert them into a format suitable for training, a set of Python scripts was created. For other preprocessing tasks, commonly available tools were used. These necessary preprocessing steps, scripts and tools are described in detail in Chapter 3. A quick overview of the most important custom scripts created for this work follows:

`readalign_to_docs.py` Converts output of `uplug-readalign` into a parallel corpus format and splits the corpus into dev/train/test sets.

`doc2context.py` Converts the output of previous script into a format suitable for a specific architecture, e.g. token-delimeted sequence of sentences for the concatenation architecture. Also performs tokenization and truecasing.

`length_hist.py` Creates a histogram of sentence lengths in a corpus.

`preprocess.sh` Runs BPE algorithm on training data for a specific model architecture.

`preprocess_wmt.sh` Runs a complete preprocessing pipeline for WMT corpus.

`context2dual.py` Converts token-delimeted corpus into multiple files, suitable for dual encoder models.

`score.sh` Scores a corpus using dual cross-entropy filtering. Used for Paracrawl preprocessing.

`hash.py` Computes hashes of sentence pairs from filtered input parallel corpora and creates a hash table with adequacy and domain scores as values. Used for Paracrawl preprocessing.

`hash_match.py` Iterates over unfiltered parallel corpus, computes a hash for each sentence pair and finds the score in the hash table created by the previous script. Used for Paracrawl preprocessing.

5.4.2 Evaluation

Evaluation procedure is described in depth in the following chapter. Again, only a brief list of scripts is presented here:

`trans_test.sh` Translates a test set with selected model, saves both the raw and postprocessed output and computes BLEU and chrF scores.

`bootstrap_par.py` Uses a bootstrap resampling to compute statistical significance of differences between two candidate translations of a test set in regard to the reference. Outputs p-value for which one system performs significantly better than the other one, and confidence intervals.

`test_all.sh` Runs the previous two scripts for selected models and saves the results.

`eval.py`	Converts a discourse test set from JSON to a parallel corpus format suitable for scoring by an NMT model and runs the preprocessing.
`get_scores_*.sh`	Scores the test set created by the previous script by a given model.
`normalize.py`	Normalize the scores obtained by the previous script by target sentence length.
`compare.py`	Compares the normalized scores and computes accuracy of a model on the discourse test set.

Chapter 6

Experiments

Experimental evaluation of different approaches to employing context in NMT is the pivotal part of my work. In the first section, some of the intricacies of MT evaluation are discussed and metrics used in this work are described. In the second part, results of the experiments are presented and analyzed. Only the most relevant results are presented here for the sake of clarity, complete overview of successful experiments is located on the accompanying data medium, along side with test and dev sets translated by some of the systems.

6.1 Tools

Preprocessing steps and tools are described in Chapter 3. Marian framework is presented in Chapter 5, so only a breif describtion of both follows.

Preprocessing Standard Moses scripts are used for tokenization, truecasing and cleaning of the data. Language filtering is performed using langid.py [39]. Finally, splitting input text into subword units to resolve the open vocabulary problem is done using subword-nmt [58]. Other preprocessing steps are performed by custom scripts.

NMT framework Marian [25], for more detailed information see Chapter 5.

Evaluation Detokenized BLEU scores [47] are computed using SacreBLEU [51]. Other metrics are calculated using custom scripts, described in the following section. To evaluate the statistical significance of the results on the test sets, bootstrap resampling [30] was used for some of the experiments. A script by Graham Neubig[1] modified for multi-process parallelism[2] was used to perform this task.

6.2 Evaluation

Evaluation of an MT system is a complicated task involving many intricacies. Ideally, human evaluators asses the quality of the translation, but even then, there is a number of issues that need to be dealt with. A good source of experience with human assessment are the Findings of Conference on Machine Translation (WMT) [7]. Since it is not feasible to

[1] https://github.com/neubig/util-scripts/blob/master/paired-bootstrap.py

[2] https://github.com/cepin19/mt_scripts/blob/master/bootstrap_par.py

evaluate every model that is developed by number of human evaluators, automated metrics are used, which are even more problematic. These metrics are usually reference-based, meaning that they compare the model-produced translation of a test set with a reference, human-made translation of the test set.

Most problems with automated metrics stem from the fact that usually a large number of correct translations of the same source sentence exists, and only one, or few of them, are available as a reference translation. This leads to penalization of systems that translate the source sentence correctly, but using different word choice than the human translated reference, since most of the metrics do not take semantics of translation, or at least synonyms, into account. To evaluate systems in terms of two commonly used automated metrics, BLEU (Bilingual Evaluation Understudy) [47] and chrF (character n-gram F-score) [50] scores, parts of training corpora were set aside to create development and test sets. The splitting was performed using `readalign_to_docs.py`, 20 documents were set aside for both dev and test sets. The scores are computed using SacreBLEU [51].

6.2.1 BLEU

BLEU score is the most widely used metric in MT. It measures overlap of tokens and token n-grams between MT generated translation and a reference translation provided by a human translator. It is a precision based metric, basically a percentage of matching n-grams (independently on the position in the sentences), with several modifications. As such, the values range from 0 for translation with no overlap with the reference, to 100 for an exact match. The calculation is done on a corpus level and it is controlled by a number of parameters:

- N – maximum n-gram length, usually $N = 4$
- ρ – brevity penalty, penalizing short translations
- smoothing type – what to do with n-grams with zero overlap count (e.g. there are no 4-grams that are the same in MT output and the reference), this issue usually does not occur on a corpus level
- case sensitivity

Main advantages of BLEU score are that only one reference translation of a test set is needed, speed and zero cost compared to a human evaluation. Additionally, the score often correlates well with a human judgement [11].

Unfortunately, many issues arise when using BLEU score. Its simplicity does not allow it to capture any semantic meaning behind the compared texts, thus a perfectly good translation can score very badly, for example because synonyms of words in the reference translation were used. Inversely, a small change in a translation may completely change the meaning of a sentence, but it may not cause a large drop in BLEU score. See for example:

Source:	It was a beautiful trip, I am glad I came along.
Reference:	Byl to nádherný výlet, je dobře, že jsem jel s vámi.
Translation 1:	Je dobře to nádherný výlet, to že jsem jel s vámi byl, je dobře, že jsem.
Translation 2:	Krásná vyjížďka, rád jsem se k vám přidal.

Depending on smoothing type, translation 1 can score 50.76 BLEU, while translation 2, even though it is much superior, scores 3.70 with the same settings.

Also, BLEU score underestimates the quality of NMT and rule-based systems compared to PBSMT systems [60]. These and other issues show that comparisons based on BLEU scores need to be taken with caution. However, it is still the most ubiquitously used metric in MT and as such it is also used as one of the main metrics in this work.

6.2.2 chrF

The smallest unit used in BLEU score computation is a word. This level of granularity may not be ideal, especially for morphologically rich languages. For example, consider sentences:

Source:	Our favorite teacher stood up and left.
Reference:	Náš oblíbený učitel vstal a odešel.
Translation 1:	Náš oblíbený učitel vstal a odešel.
Translation 2:	Naše oblíbená učitelka vstala a odešla.

Both translations may be correct, depending on the teacher's gender. However, translation 1 would obtain BLEU score of 100, while translation 2 would get a very low BLEU score (7.68). To mitigate this issue, several character-level metrics were presented. One of the recent ones, showing a good correlation with human assessment of translation quality, is chrF – character n-gram F-score, with n usually set to 6. The formula is following:

$$\mathrm{chrF}\,\beta = \left(1+\beta^2\right)\frac{\mathrm{chrP}\cdot\mathrm{chrR}}{\beta^2\cdot\mathrm{chrP}+\mathrm{chrR}}$$

chrP and chrR are character n-gram precision and recall, averaged over all n-grams. β determines how many times is recall more important than precision. In the original paper, $\beta = 3$ was shown to have highest correlation with human judgement. ChrF3 of the first translation equals to 1.0, and chrF3 of the second translation equals to 0.52 – the difference is much more realistic than in case of the BLEU scores (100 and 7.68).

ChrF is a useful metric for languages with high morphological variance, like French or Czech. Also, NMT models often operate on subword level, and thus evaluating on smaller language units may be beneficial, since, as mentioned earlier, word-level metrics like BLEU seem to underestimate NMT translation quality.

6.2.3 Discourse test set

For more targeted evaluation of inter-sentential phenomena, an approach used by Bawden et al. [5] was adopted. The authors created a manual contrastive test sets to quantify a machine translation system accuracy in translating coreference and coherence/cohesion phenomena. The set comprises of source sentences and both correct and incorrect translation. NMT model is used to score both translation in terms of cross-entropy. The translation with higher probability is chosen and overall accuracy is calculated. The test set is balanced so that any system without employing context scores 50%. In disambiguation part, which is used in this paper, there is one current source sentence, two possible previous source sentences and two possible translations - each one correct in one of the contexts. For example:

Context 1:	We went to the cliffs to watch our favorite seal in the sea.
Context 2:	His house was sealed by the police because of the crime investigation.
Source:	When we have seen the **seal**, we went back home.
Translation 1:	Když jsme toho **lachtana** uviděli, šli jsme domů.
Translation 2:	Když jsme tu **pečeť** uviděli, šli jsme domů.

For both contexts, correct and incorrect translations are scored by the model and the more probable one is chosen. Final accuracy is computed based on how many times the correct translation was preferred over the incorrect one. Since the test set contains both possible correct combinations paired with the wrong ones, a system without any knowledge of the previous context will always score 50% (as it will choose the same target sentence as more probable both times, the cross-entropy score will be the same for both contexts). For English-French, the original dataset was used[3], for English-Czech, relevant parts were translated and combined with newly created examples[4].

6.2.4 Model naming conventions

Different configurations of input and output are labeled src$N$tgt$M$ to tgt$K$, where N and M are counts of previous source and target sentences concatenated to the input, and K denotes how many previous target sentences are to be generated on the output.

Thus, *src0tgt0 to tgt0* is an ordinary, vanilla NMT architecture, without any context influence, *src1tgt0 to tgt0* means that one previous source sentence is concatenated to the input, *src0tgt1 to tgt0* means that one previous target sentence (i.e. translation of previous source sentence) is concatenated to the input and so on. For systems generating more than one target sentence, for example *src1tgt0 to tgt1*, the target side of the training data is preprocessed in the same way, this means that the model is trained to generate multiple sentences.

For systems employing target context on source side, a slight issue appears. The target context (translations) must be obtained somehow. It is cumbersome to generate the previous sentence translation using the same model – usually, the translations are done in batches which consist of subsequent sentences that are translated simultaneously. Sequential dependencies between target sides of the sentences would prohibit processing them in one batch, which could be solved by interleaving sentences from different documents to create a batch. However, this is possible only under a big workload, when lots of independent documents are translated in parallel.

Instead of using target context created by the model itself, I opted for two different configurations. First, *src1tgt1ref to tgt0* means that the reference sentences are used as a target context. Such setting is similar to post-editing scenario – human translator approves or corrects a previous sentence translation, which is subsequently used as an input for translating the next sentence.

In the second configuration, *src1tgt1mt to tgt0*, the target context sentences are generated beforehand by a model using only a source context (*src1tgt0 to tgt0*). This configuration is utilizable when translation of the whole document at once is necessary. The total

[3] https://github.com/rbawden/discourse-mt-test-sets

[4] https://github.com/cepin19/discourse-test-set

sum of computation time is larger, each sentence is translated twice, but the translation can be parallelized (in two steps).

Finally, for the newly implemented models, *ref1src0 to tgt0* configuration was also used as a sanity check, meaning that ther reference translation was used as a context. This was done to see whether the model is capable to use information provided by the context encoder – if that is the case, BLEU scores near 100 should be expected.

6.3 Results

In this section, experimental results are presented and discussed. Only the most relevant results are presented here, a more complete list can be found either on the attached CD, or at https://github.com/cepin19/dp.

6.3.1 Concatenation

Adding context to **RNN** through concatenation hurts the translation performance, as presented in Table 6.1. This is in line with observations made by Agrawal et al. [1], where the authors obtained similar result on ISWLT 2017 English->Italian data set, consisting of transcribed TED Talks. For configurations *src1tgt0 to tgt0* and *src1tgt0 to tgt1*, BLEU drops of 1.8 and 2.8 points, respectively, were observed. Arguably because even though there are gating mechanisms employed, RNNs suffer from loss of signal in very long-range dependencies.

In Bawden et al. [5], the results are quite different. Concatenating only a previous source sentence to the input decreased the performance on most genres (the authors trained and evaluated their systems on OpenSubtitles, and used test sets from different movie genres). When using *src1tgt0 to tgt1* configuration, i.e. adding previous target sentence to the output, BLEU score surpassed baseline in all test sets. One of the reasons may be that OpenSubtitles contain much shorter lines on average, and the models do not suffer from weak long-range dependencies. Since RNN networks perform worse than the Transformer model in most cases [33], and take longer to train, more advanced experiments were performed on the Transformer model only.

metric	src0tgt0 to tgt0	src1tgt0 to tgt0	src1tgt1 to tgt0	src1tgt1 to tgt1
BLEU	**29.07**	28.88	27.82	27.33
disambig	50%	50.7%	-	-

Table 6.1: Results of concatenation experiments with **RNN**, **English to Czech, Europarl**, average of three runs, dev set

metric	src0tgt0 to tgt0	src1tgt0 to tgt0	src1tgt0 to tgt1
BLEU	34.58	**34.99**	34.05

Table 6.2: Results of concatenation experiments with **Transformer**, **English to French, Europarl**, **dev** set

metric	src0tgt0 to tgt0	src1tgt0 to tgt0	src1tgt1ref to tgt0	src2tgt0 to tgt0
BLEU	27.77	27.93	**28.4**	27.97
disambig	50%	**60.4%**	52.8%	-

Table 6.3: Results of concatenation experiments with Transformer, **English to Czech, OpenSubtitles**, **dev** set BLEU score and accuracy on discourse test set

context type	architecture	n	real c.	random c.	null c.	$\Delta_{real,random}$	disambig
0,0 to 0	baseline	0.6	34.92	-	-	-	50%
0,0 to 0	baseline	opt	36.38	-	-	-	50%
1,0 to 0	concat	0.6	35.36	34.85	35.35	0.51	**62.4%**
1,0 to 0	concat	opt	**36.60**	35.97	36.35	0.63	**62.4%**
1,0 to 0	DE	0.6	35.36	35.14	1.41	0.22	50.6%
1,0 to 0	DE	opt	36.47	36.32	1.44	0.15	50.6%
1,0 to 0	DE, shared	0.6	34.98	34.32	1.38	0.65	50.6%
1,0 to 0	DE, shared	opt	36.33	35.64	1.39	0.69	50.6%
1,0 to 0	DE, shared + tok	0.6	35.24	34.93	1.41	0.31	51.8%
1,0 to 0	DE, shared + tok	opt	36.42	36.15	1.43	0.27	51.8%
1,0 to 0	CE	0.6	34.83	34.65	1.37	0.18	51.2%
1,0 to 0	CE	opt	36.41	36.20	1.42	0.21	51.2%
1,0 to 0	Voita	0.6	34.82	34.15	1.88	0.15	52.4%
1,1ref to 0	concat	0.6	37.12	20.28	33.93	16.84	58.3%
1,1ref to 0	concat	opt	**37.44**	20.92	35.14	16.52	58.3%
1,1mt to 0	concat	0.6	35.10	-	-	-	58.3%
1,1mt to 0	concat	opt	35.90	-	-	-	58.3%

Table 6.4: Results for models trained on **English to French, OpenSubtitles**, Transformer model, **dev** set. First column shows type of context used. The first number stands for previous source sentences added to input, second one for previous target sentences added to input and the third one is number of additional previous target sentences generated by the model, so for example 1,0 to 0 equals to model denoted *src1tgt0 to tgt0* elsewhere in the text. Second column is the model architecture, DE stands for dual encoder, CE denotes the context encoder model. For dual encoder, *shared* means that all weights in the two encoders are shared, in *shared+tok* strategy, a special token is added to the start of the previous sentence – since all the layers are shared, the encoder would otherwise be unable to distinguish between context and source sentence during the computation, and that may not be optimal. Third column is the length normalization coefficient – *opt* means an optimal value found by search over possibilities within a given range, see paragraph Length normalization. In columns number 4, 5, and 6, BLEU scores depending on whether real, random, or empty context sentences were used, are shown. Next column shows difference between real and random context BLEU scores. Finally, in the last column, accuracy on disambiguation part of contrastive discourse test set is presented.

As presented in Tables 6.2, 6.3, 6.4, 6.5 and 6.6, the performance of the **Transformer** model does not degrade when concatenating the sentences at the input, confirming the

context type	architecture	n	r	BLEU	BLEU interval	chrF1	chrF3	p-value
0,0 to 0	baseline	0.6	1.018	31.59	[0.311, 0.321]	0.5036	0.5062	-
1,0 to 0	concat	0.6	1.019	**32.07**	[0.315, 0.326]	0.5080	0.5114	**0.000**
1,0 to 0	DE	0.6	1.010	31.62	[0.311, 0.322]	0.5013	0.5028	0.411
1,0 to 0	DE, shrd	0.6	1.005	31.47	[0.309, 0.320]	0.5007	0.5013	-0.189
1,0 to 0	DE, shrd+tok	0.6	1.015	31.59	[0.310, 0.321]	0.5032	0.5058	0.487
1,0 to 0	CE	0.6	1.020	31.50	[0.310, 0.320]	0.5038	0.5069	-0.104
1,1ref to 0	concat	0.6	1.018	**33.14**	[0.326, 0.337]	0.5181	0.5222	**0.000**
1,1mt to 0	concat	0.6	1.033	31.15	[0.306, 0.317]	0.5022	0.5093	-0.001

context type	architecture	n	r	BLEU	BLEU interval	chrF1	chrF3	p-value
0,0 to 0	baseline	2.9	1.122	29.31	[0.288, 0.298]	0.4977	0.5193	-
1,0 to 0	concat	1.7	1.117	**29.81**	[0.293, 0.303]	0.5016	0.5232	**0.000**
1,0 to 0	DE	2.0	1.114	29.25	[0.287, 0.298]	0.4962	0.5166	-0.308
1,0 to 0	DE, shrd	2.0	1.114	29.01	[0.285, 0.295]	0.4953	0.5157	-0.004
1,0 to 0	DE, shrd+tok	1.9	1.108	**29.54**	[0.290, 0.301]	0.4979	0.5176	**0.023**
1,0 to 0	CE	0.6	1.120	29.38	[0.310, 0.320]	0.4983	0.5198	-0.104
1,1ref to 0	concat	1.0	1.083	**31.47**	[0.309, 0.320]	0.5134	0.5306	**0.000**
1,1mt to 0	concat	1.5	1.113	29.45	[0.289, 0.300]	0.4979	0.5202	0.158

Table 6.5: Results for models trained on **English-French OpenSubtitles**, Transformer model, **test** set. Description of the columns from Table 6.4 applies. In the top table, n is set to 0.6. The length normalization constant for the bottom table the one performing the best on the dev set. Values of r represent token length ratio of the MT translation and the reference. BLEU score column shows the score for the whole test set, BLEU interval column shows 95% confidence interval for BLEU score, using bootstrap resampling with *n=10000*. The last column presents the p-value for which the model performs significantly better than the baseline (or worse, in case of negative values). Scores of models outperforming the baseline for $p > 0.05$ are written in bold.

assumption that the Transformer is better equipped to deal with longer sequences and longer dependencies between input symbols than RNNs. Concatenation was in most cases the only architecture that yielded significant improvements. For English-Czech Europarl dataset (Table 6.6), all of the methods failed to significantly improve upon the baseline. The results in English-French OpenSubtitles (Table 6.4 and Figure 6.1) are more diverse, suggesting that this type of corpus may be more suitable for incorporating context information. The configuration that achieved the largest BLEU improvement was *src1tgt1 to tgt0* on English-French dataset.

Results presented in Tables 6.5 show that the improvements for English-French OpenSubtitles, observed on dev set, can also be seen when evaluating on the test set. The systems were also evaluated in terms of chrF1 and chrF3 on the test set. The same conclusions as for BLEU scores hold true for chrF scores – the concatenation systems perform the best. The systems in the bottom part of Table 6.5, with n tuned on the dev set (explained later, Subsection 6.3.7), perform worse than the ones in the top table in BLEU and chrF,

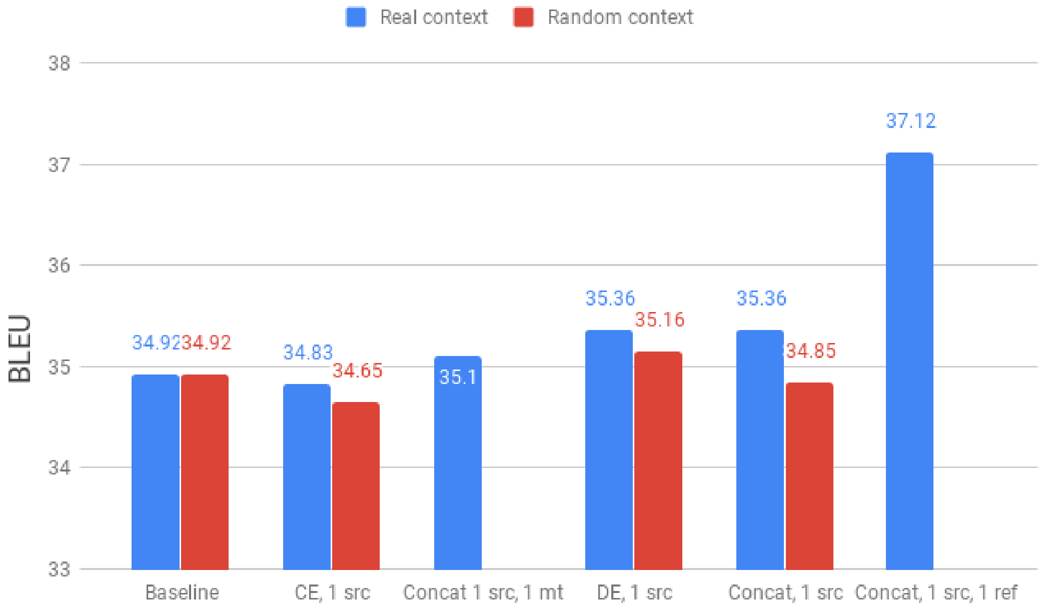


Figure 6.1: Results of some of the notable models on **English-French OpenSubtitles dev** set. Complete results are presented in Table 6.4.

but better in chrF3 metric. This is caused by the fact that these systems generate longer sentences (as apparent from the r column) and chrF3 score assigns 3 times more weight to recall than to precision. To asses statistical significance of improvements over the baseline, bootstrap resampling was used. The concatenation systems performed significantly better than the baseline, for $n = 10000$ and $p > 0.05$.

During the experiments, I ran into an issue with Transformer batch sizes and the framework I used. Transformer training is very sensitive to minibatch size [49] and a memory of one GPU is not sufficient to store an optimally sized minibatch. I only had limited access to GPUs, and sometimes only one GPU was available for experimenting. Luckily, Marian has a feature called optimizer delay, which accumulates gradients for n minibatches before updating the weights, effectively scaling the minibatch size by factor of n. Less luckily, this feature was broken until a recent commit[5], which I found out during the experiments, since some of the runs with a same config were significantly worse the others, so I had to rerun the experiments. The lesson here is to always check whether larger batch size can bring an improvement when training the Transformer.

Largest difference in BLEU scores was observed in *src1tgt1ref to tgt0* concatenation on English-French OpenSubtitle dataset, i.e. base Transformer model with one previous source and one previous target sentence concatenated to the input sentence. The target context sentences in this scenario are taken from the reference data. When using model-generated previous target context sentences (*src1tgt1mt to tgt0*), created by the *src1tgt0 to tgt0* model, the performance is significantly worse, probably due to error propagation.

This configuration was not investigated further, since the dependence on previous target sentence is preventing parallelization of translation using batching, thus making this

[5] https://github.com/marian-nmt/marian/issues/259

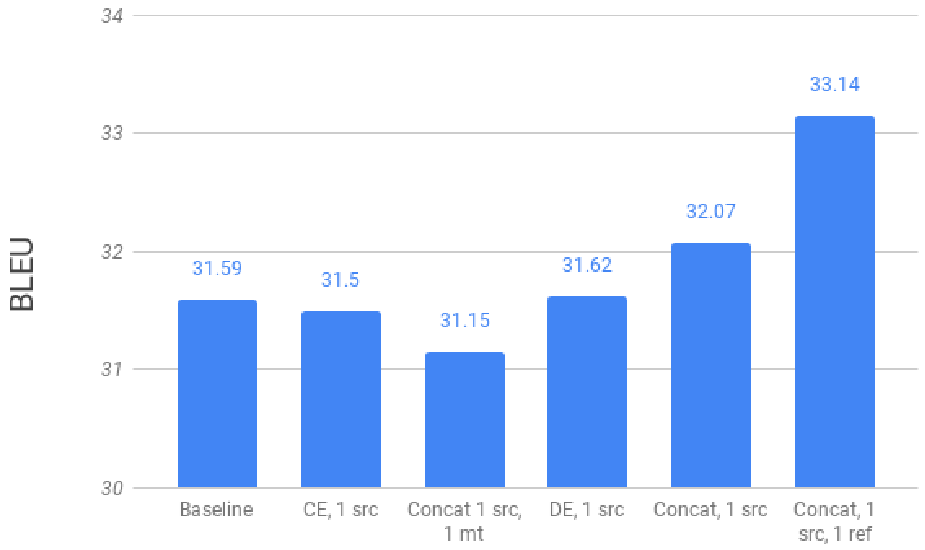


Figure 6.2: Results of some of the notable models on **English-French OpenSubtitles test** set. Complete results are presented in Table 6.5.

approach impractical. However, the configuration with previous reference sentences is somewhat similar to post-editing scenario, where a human translator sequentially corrects translations made by an MT system, so the corrections made in previous sentences can be used to improve translation of the future sentences.

6.3.2 Dual encoder

Slight gains in BLEU score were obtained by a simple dual encoder models in some of the configurations, see Tables 6.4, 6.5 and 6.6. However, these gains are probably caused by the larger number of learnable parameters of the model, rather than by a correct context utilization. This issue is discussed later in the text, in Subsection 6.3.6.

6.3.3 Context encoder

First of the models that I implemented in Marian was document-level Transformer by Zhang et al. [73], denoted *context encoder* or *CE* in the tables. The original implementation was evaluated on English-Czech Europarl and English-French OpenSubtitles. The results are presented in Tables 6.7 and 6.8. For Europarl, a slight improvement in BLEU (0.3) was obtained on the dev set. After evaluating my implementation on the same data, a similar gain (0.2) was observed. For English-French OpenSubtitles, a drop of about 0.2 BLEU point was observed, again similar to my reimplementation. I was not able to evaluate the original implementation on the discourse test set, since it does not provide a way to score a translation.

As a sanity check, to see if the implemented model is able to use input provided in the additional encoder at all, the reference translation was used as a context sentence. After a few thousand updates, BLEU score reached values around 98, proving that the model is indeed able to learn to use the context encoder.

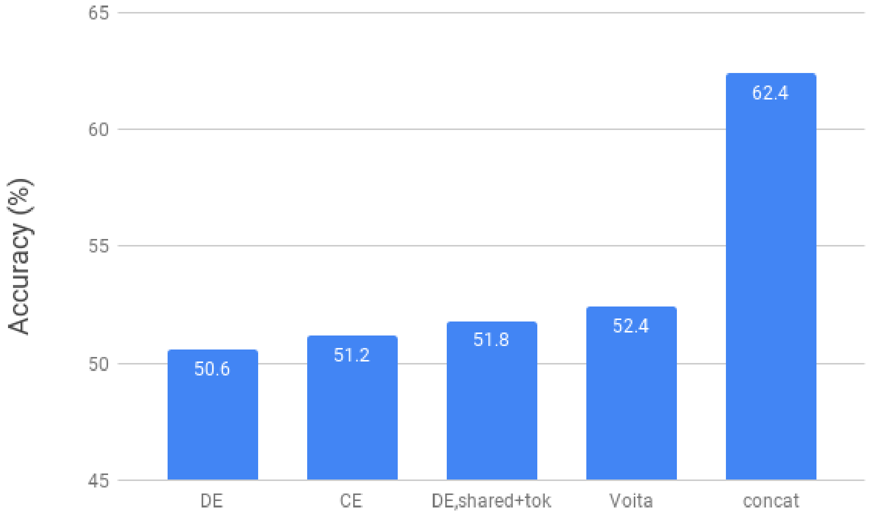


Figure 6.3: Discourse test set accuracy of models trained on **English-French OpenSubtitles**. Simple concatenation model outperforms other architectures by a large margin, similarily as in BLEU score evaluation.

One of the advantages of this model is that it can be pretrained as a normal Transformer on sentence-level corpus and only the weights of the additional components can then be fine-tuned on a small, document-level corpus. In the experiments on Europarl (Table 6.6) and OpenSubtitles (Table 6.4), the models were trained on identical corpora in both phases (in the first one without the context level information), so it is expected that there is no observable gain from pretraining, other than quicker convergence of training in the second phase (much fewer parameters have to be learned).

To asses the effect of pretraining and to see how the model performs on a larger scale, an additional experiment on WMT data was performed. The baseline models were trained on WMT19 English-Czech corpora and Paracrawl, preprocessed as described in Chapter 3 (including the special preprocessing for Paracrawl). Two models were trained, referred to as *weak* and *strong* in Table 6.10. *Weak* model is a Transformer-base model trained solely on the parallel data. *Strong* model is a Transformer-big, trained on the parallel data and backtranslated Czech News Commentary 2007-2018 datasets. For both models, context encoder architecture performs worse than the baseline.

In neither of the experiments was I able to gain any significant improvement with the context encoder model. I tried to tweak hyperparameters like number of layers of the context encoder, different gating function (Table 6.9) or not pretraining the model, but none of the changes had the desired impact. For the Europarl and OpenSub datasets, models were alse trained using the original implementation for comparison, and similar results were observed. These results show that the performance of the proposed architecture is strongly dependent on the dataset used, since in the original paper authors observed significant improvements on a different dataset.

Model	**dev set BLEU**	**test set BLEU**
Baseline	29.6	33.0
src1tgt0 to tgt0	29.9	33.2
Marian baseline	30.3	33.4
Marian src1tgt0 to tgt0	30.5	33.5

Table 6.7: Comparison of results of original implementation of context encoder by Zhang et al. [73] and reimplementation in Marian, English-Czech Europarl

.

Model	**dev set BLEU**	**test set BLEU**
Baseline	33.0	30.1
src1tgt0 to tgt0	32.8	30.0
Marian baseline	34.9	31.6
Marian src1tgt0 to tgt0	34.8	31.5

Table 6.8: Comparison of results of original implementation of context encoder by Zhang et al. [73] and reimplementation in Marian, English-French OpenSubtitles

.

6.3.4 Context-aware Transformer

The second model that was implemented in this work is the context-aware Transformer by Voita et al. [69]. In a sanity check with reference sentence as a context, the model obtained BLEU score of 99. Next, the architecture was evaluated on English-French OpenSubtitles and English-Czech Europarl. In both settings, the model performed worse than the baseline. Some evidence of a correct context usage was observed on the discourse test set for the English-French OpenSubtiles-trained model, see Table 6.4.

No reference implementation of this model is available, so the poor performance may be caused by an implementation error. However, in a recent paper by Jean and Cho [20], this model also fails to improve upon the baseline. I performed sanity checks to see whether the model is able to use information from the second encoder. These checks suggest that the model can indeed use the additional information and that the implementation may be correct.

6.3.5 Discourse test set

Some of the models were also evaluated in terms of accuracy on the discourse test sets described earlier, and the results are presented in *disambig* columns in corresponding tables. The only significant gains, again, were obtained by concatenation architecture, trained on OpenSubtitles. Europarl models did not perform well on this test set, partially due to domain mismatch (test set examples are much closer to sentences found in subtitles than to the ones found in Europarl) and partially due to bad overall context utilization in Europarl-trained models. Only disambiguation based on one preceding source sentence was evaluated. Examples of correct and incorrect disambiguation performed by *src1tgt0 to tgt0* concatenation models follow:

Context sentence	Source sentence	Chosen translation
But who's ever going to stand up to her ?	If anyone can take her on, it's you.	Jestli se jí někdo může postavit, jsi to ty.
But in such a sorry state, who's going to look after her ?	if anyone can take her on, it's you.	Jestli se jí někdo může ujmout, jsi to ty.
Are we having the chicken tonight?	I couldn't bring myself to pluck it.	Nedokázala jsem se přinutit ho oškubat.
That hair's still there, you know.	I couldn't bring myself to pluck it.	Nedokázala jsem se přinutit ho vytrhnout.
We need to boil the pasta in something	Do you have some pot ?	Máš nějakou trávu?
I really need to get high right now.	Do you have some pot?	Máš nějakou trávu?
The weather's starting to get hot.	Too hot for me.	Na mne je moc horko.
The chicken curry's meant to be good.	Too hot for me.	Na mne je moc horko.
The captain said we're sailing to France.	Which port does he want?	Il veut quel port?
They've finished the main course and are asking for dessert.	Which port does he want?	Il veut quel porto?
So anything new on the suspects?	I found a match!	J'ai trouvé une correspondance!
My lighter isn't working...	I found a match!	J'ai trouvé une allumette!
Evacuate immediately.	Is this not a drill?	Ce n'est pas un exercice?
I need you to go and find me a drill.	Is this not a drill?	Ce n'est pas un exercice?

Oh, I do like your... the things holding up your trousers.	They're braces.	C'est un appareil dentaire.
Oh, what a pretty smile, despite the thing on your teeth.	They're braces.	C'est un appareil dentaire.

Dual encoder models, despite the slight gains in BLEU on the OpenSubtitles corpus, do not seem to utilize context information well – accuracy for different model checkpoints fluctuated between 48.5 and 51.5 % for a simple dual encoder model. Better utilization of context can be observed when the encoders share learnable parameters, but the accuracy is still worse than the accracy of concatenation models by a large margin.

This suggests that the BLEU gains for dual encoder models are caused by other effects than a successful context utilization. For dual encoder and context encoder configurations, the BLEU gains are observed probably mainly due to increased number of parameters in comparison to the baseline model - there are more attention layers and subsequent feedforward layers, which in theory should serve to incorporate the context information, but their main contribution in reality is presumably improving the representation of current source sentence.

6.3.6 Adversary context

Several models were also evaluated with a random context as an input, instead of a real one. Quite surprisingly, the results were not much worse with the random context sentences, especially for Europarl corpus, as presented in Tables 6.6 and 6.4. This, along with results on the discourse test set, shows that the models do not depend on context information too much. As mentioned in last paragraph, the BLEU gains over the baseline for multi encoder models can be explained by increased number of parameters of the model.

For concatenation configuration, this is not true, model architectures are exactly the same regardless whether the context is used or not. However, on English-Czech Europarl corpus (see Table 6.6) an improvement over the baseline (29.6 BLEU) can be observed for concatenation system, even when random context is used (30.3 BLEU). Maximum source sentence length for training is set differently for baseline and concatenation models, which seems to be the issue.

6.3.7 Maximum source sentence length

For Europarl corpus, maximum length of the source sentence was set to 80 subwords for no context, multiplied by the number of context sentences for concatenation models. Longer sentences were omitted from the training. Since it is not probable that two exceedingly long sentences follow each other in the corpus, concatenation models had chance to train on these long sentences, while the baseline model excluded them. I assumed it will not hurt the performance too much, based on a sentence length analysis, only 1.2 % of the source sentences were longer 80 subwords in English-Czech Europarl. As it turned out, this assumption was wrong.

When trained with maximum input length of 160, baseline model performs the same, or better, as the concatenation models, reaching BLEU score of 30.3 on English-Czech

Europarl dev set. This does hold true for Europarl, on OpenSubtitles, I did not observe this problem and even for a larger length limit, improvements in BLEU score were observed on OpenSubtiles. This suggests that there is more to gain using context on OpenSubtitles dataset than on Europarl, or that different techniques need to be used for different datasets.

6.3.8 Length normalization

Usually, in NMT, beam search is used to select the best sentence translation from hypotheses generated by the model. Beam search in NMT has two hyperparameters – beam size and length normalization constant n. Without length normalization, probabilities of each word along the beam are summed up and then the best overall score (log-likelihood) is chosen. Usually, this results in preference for shorter sentences, since less total tokens in the output will probably mean lower (better) score. To mitigate this issue, which often harms translation quality, the final summed score is divided by number of output tokens l to the power of n: l^n. However, n has to be chosen empirically since its optimal value varies from language to language, dataset to dataset, and model to model. Popular choice is 0.6, which is the default used in experiments in this paper, if not stated otherwise.

However, for some of the models, optimal n was determined on the dev set by search in interval 0.4-3.0 (with step 0.1). Results are shown in Table 6.4. Optimal n was always much higher than 0.6, usually in range 1.5-2.5. Also, it is different for each model, so probably the most fair way to compare the models is to choose optimal n on the development set for each model and then compare the test set scores with these parameters, as in Table 6.5. Beam size and length normalization value are not independent of each other - an ideal solution would be to run a grid search along these two parameters, which was not done due to computing restraints - beam size was set to 6 for all the experiments.

The optimal n found by the search on the dev set was also tried out when translating the test set. As presented in Table 6.5, models generally performed worse with the found n than with n=*0.6*. This observation confirms that this value strongly depends on the test set used.

6.3.9 Manual inspection

During the development of a new system, automated metrics are necessary to compare the candidates. However, to see if the system really improves the translations in the way that the metrics suggest, manual inspection of the translated sentences is a very valuable tool. Examples of passages of the text where the context seems to be used appropriately follow. Of course, these are only a few cherry-picked examples and do not say much about the system quality. First set comes from English-Czech OpenSubtitles dev set and a concatenation system with two previous source sentences as a context is used:

Source	src0tgt0 to tgt0	src2tgt0 to tgt0
Did he tell me to buy front leg? Or back leg?	Řekl mi, ať si koupím přední nohu? Nebo zadní noha?	Řekl mi, abych koupil přední nohu? Nebo zadní nohu?

Let go of her hand. - Go back and do your dance. You can't let go?	Pusť ji. - Vrať se a zatancuj si. Nemůžeš to nechat být?	Pusť jí ruku. - Vrať se a zatancuj si. Nemůžeš ji pustit?
Do you think you can destroy it with just that power? I was planning to do this even without it .	Myslíš, že ji můžeš zničit jen s takovou silou? Chtěl jsem to udělat i bez něj.	Myslíš, že ji můžeš zničit jen s takovou mocí? Plánoval jsem to udělat i bez ní.

Another set of examples from the discourse test set, using a model train on English-Czech OpenSubtitles for the translations. Only translation of the second sentence is provided, since that is the interesting sentence with an ambiguous translation:

Source	src0tgt0 to tgt0	src1tgt0 to tgt0
We've been thinking about it and we'd like to throw a ball this weekend. A ball!	Míč!	Ples!
The new guy is a bit rude. And dim.	A temný.	A tupý.
Okay, now you want to turn right. No, it's my right!	Ne, je to moje právo!	Ne, to je moje doprava!
The makeup looks good. We do silver nails too.	Taky děláme stříbrné hřebíky.	Děláme i stříbrné nehty.
You've gone and broken the teacup! I don't give a damn about the cup.	Na ten pohár kašlu.	Kašlu na ten hrnek.
I should probably be off now. But this is your place.	Ale tohle je tvoje místo.	Ale tohle je tvůj byt.
The pizza is in the oven, but there's still some dough left. I've never seen so much dough!	Nikdy jsem neviděl tolik prachů!	Nikdy jsem neviděl tolik těsta!
We could use a brush to detangle the hair. I don't think we should use a brush.	Nemyslím , že bychom měli používat štětec.	Nemyslím, že bychom měli použít kartáč.

On the other hand, many examples where the context was less successfully employed can be found among the sentences from the discourse test set. Some of the translation do not show any traces of context influence, while others seem to be at least half-correct. The last sentence is translated correctly without the context, and incorrectly by the context model::

Source	src0tgt0 to tgt0	src1tgt0 to tgt0

Do you need anything for the game, son? Like a bat?	Jako netopýr?	Třeba netopýra?
Where did you say the chicken was? Over there by the pen.	Támhle u pera.	Támhle u pera.
We should be getting back to the church. The minister will be getting worried.	Ministr bude mít obavy.	Ministr bude mít obavy.
Could it be anything serious, Doctor? We'll have to get rid of that mole.	Musíme se zbavit toho krtka.	Musíme se zbavit toho krtka.
The verdict isn't great. What is the final sentence?	Jaká je poslední věta?	Jaká je poslední věta?
Did you give her a slice of tart? No, it was a piece of cake.	Ne, byla to hračka.	Ne, byla to hračka.
He crawled into that cannon himself and ask me to do it. Yes, but I still don't think you should fire him.	Ano, ale stejně si nemyslím, že bys ho měl vyhodit.	Ano, ale stejně si nemyslím, že bys ho měl vyhodit.
But you need good nails to play the guitar. It might be easier with a pick.	S trsátkem by to bylo jednodušší.	S krumpáčem by to bylo jednodušší.

context type	architecture	len	n	real c.	random c.	null c.	disambig
0,0 to 0	baseline	80	0.6	29.6	-	-	50%
0,0 to 0	baseline	160	0.6	30.3	-	-	50%
1,0 to 0	concat	160	0.6	30.3	30.3	29.4	51.8%
1,0 to 0	dual encoder	80	0.6	30.0	30.0	1.9	50.5%
1,0 to 0	CE	80	0.6	29.8	-	-	-
1,0 to 0	CE, +gate	80	0.6	29.9	-	-	-
1,0 to 0	CE, +gate, pretrain	80	0.6	30.0		-	-
1,0 to 0	CE, +gate, pretrain	160	0.6	**30.5**	30.4	0.1	**52.8%**
1,0 to 0	CE, switch	160	0.6	29.6	-	-	-
1,0 to 0	Voita	160	0.6	29.8	-	-	-
1,0 to 0	Voita, switch	160	0.6	29.3	-	-	-
1,0 to 1, 1st sent	concat	160	0.6	30.0	-	-	-
1,0 to 1, 2nd sent	concat	160	0.6	29.8	-	-	-
1,0 to 1, 1st sent	concat	160	1.9	30.2	-	-	-
1,0 to 1, 2nd sent	concat	160	1.9	30.1	-	-	-
1,1 to 0	concat	240	0.6	29.97	-	-	-

Table 6.6: Results for models trained on **English-Czech Europarl**, Transformer model, **dev** set. For detailed description of the columns, see previous table. The additional *len* column shows maximal sentence length in subwords for training, see Subsection 6.3.7. (Maximum source sentence length) for further discussion of this issue. *CE* denotes the context encoder model, *+gate* is the same model improved with sigmoid gate to filter the influence of context, *pretrain* means that the model was pretrained on the same corpus without context information. For 1,0 to 1 context type, the model is trained to generate not only the current target sentence, but also the previous one, separated by a special token. 2nd sentence score is obtained by striping off the first (previous) target sentence and calculating BLEU on dev set, whereas 1st sentence score is obtained by cutting off the second (current) target sentence and computing BLEU of the first target sentence on reference set that is shifted accordingly by one sentence. For *CE, switch* and *Voita, switch* architectures , inputs of the encoders were switched – the context encoder encoded the source sentence and the source encoder encoded the context sentence. This configuration serves as a sanity check – showing that the model can utilize the context encoder, since it is able to translate the source sentence when its encoded by the context encoder.

# context enc. layers	**gating function**	**pretraining**	**freezing**	**BLEU**
non-contextual baseline	-	-	-	**35.92**
1	original	yes	yes	35.83
1	original	no	no	35.71
1	original	yes	no	35.85
1	none	yes	yes	35.65
1	modified	yes	yes	35.79
2	original	yes	yes	35.66
6	original	yes	yes	35.81

Table 6.9: Results of context encoder model for different number of context encoder layers and different gating functions. Column titled *pretraining* shows whether the model was pretrained on sentence-level corpus, and the next column shows whether the pretrained weights were frozen during the fine tuning on document-level corpus. No significant differences are observed for any of the parameters, none of the configurations outperformed the non-contextual baseline. **English to Czech, OpenSubtitles, dev set** BLEU score.

model	BLEU	+Context encoder	+gating	Discourse test set
weak baseline	25.28 (26.64)	24.93 (26.25)	24.97 (26.33)	52.08%
strong baseline	28.3 (29.95)	27.42 (29.03)	27.25 (29.12)	51.5%

Table 6.10: Results for models trained on **English-Czech WMT** data, newstest2018 test set (newstest2016 used as a dev set, results in parenthesis). Weak baseline is a Transformer-base model, strong baseline is Transformer-big with backtranslated data

.

Chapter 7

Conclusions

This final chapter first presents some of the easily achievable improvements and possible future research directions. In the second part, outcomes of this thesis are summarized and conclusions are presented.

7.1 Short term goals

Even though I aimed to do an extensive exploration of current context-aware architectures, many possible configurations were left untested. Several new experiments can be performed with the same, or only slightly modified models. I plan to evaluate the modifications presented in the next few paragraphs, since I intend to implement the best performing architecture in practice.

7.1.1 Forward context

In all of the experiments, only previous context was used. However, even a future context may hold some information valuable for translation of a current sentence. I plan to run experiments with the same model architectures, using forward context together with the backward one.

7.1.2 Context-aware learning

The results of experiments using random context sentences suggest that the context-aware models do not depend on context information too much – a recent paper by Jean and Cho [20], which is described in Chapter 2, confirms this observation. The authors propose a model-independent modification of the cross-entropy loss function, which is aimed to make the model more sensitive to the context. Since this algorithm can be used with any neural network MT architecture, it is an interesting future research direction, which I plan to try out before using the models in production environment.

7.1.3 Longer context window

All of the approaches presented in this work assume only a small context window of several surrounding sentences. Helpful information for the translation might however lie outside this window. A sparse, hierarchical attention mechanism was presented in a recent paper by Maruf et al. [42]. This mechanism first focuses on relevant sentences in the text, and then on the most relevant tokens in these sentences, to create a document-level context vector. This additional input can be incorporated into the translation model, extending the possible context size.

7.1.4 More detailed manual inspection

I did perform manual analysis consisting of checking the systems translations and looking for examples of correct and incorrect context usage. However, I did not analyze the translations profoundly to see exactly which types of phenomena are easy or hard to address for the model. Also, it might be interesting to look at the attention weights of words in the context sentences when the ambiguous parts of the source sentence are processed, to see for example if the model performs some kind of coreference resolution.

7.1.5 Integration with computer-aided translation system

Even though the results may not be spectacular, some improvements were observed. I plan to integrate the context-aware models into a CAT system used by human translators. I also intend to explore target context incorporation in more detail, since it might be beneficial to use sentences already corrected by the human translator as an additional input for the machine translation.

7.2 Long term goals

Surrounding sentences form only a small subset of possible context which can help with translation of the input. Human translators need to have general world knowledge and domain knowledge to translate a text perfectly. Of course, a perfect translation model would need a perfect artificial general intelligence, which is not near by any means. However, it may be possible to at least partially embed some kind of external world knowledge into an NMT model. A recent work in this area is a paper by Moussallem et al. [45], where the authors use knowledge graphs as an additional input into an NMT system, resulting in significant improvement over the baseline model.

Another way to incorporate some type of world knowledge into an NMT model is to use approaches known from question answering systems, i.e. to use an information retrieval system to find a passages of text in a large corpus, which are relevant to the translation of an input. Novel approaches using sparse attention [42] or adaptive attention spans [62] could enable processing of long passages of text and incorporation of relevant information into self-attention based models.

7.3 Summary

This work summarizes current state-of-the-art in dealing with extra-sentential context in NMT, focusing mostly on the Transformer model. Some of the simpler architectures were evaluated and compared both in terms of general translation quality and evaluation focused on discourse phenomena. Two context-aware architectures were implemented in a framework suitable for production systems. However, both of them have failed to bring any significant improvements over the baseline. The results have shown that a very carefully controlled experiments must be performed to asses the effect of context in NMT. Small changes in parameters like maximum sentence length, length normalization, or batch size, led to differences in automated metrics, which were in several cases larger than differences caused by context incorporation.

A hand made discourse test set for English to Czech translation was created. The experiments have shown that either the implemented context-aware models are not very efficient at employing context, or there is a very little to gain in automated translation quality metrics by using context models, even though this varies by the dataset used. Evaluation focused on specific discourse phenomena was performed on a hand-made test set developed for this purpose. A limited evidence of correct context usage was observed on this test set for several of the models. Manual inspection of the translations generated by a context-aware model was also performed and many cases where the translation was improved by the context knowledge were found.

The best performing models (the only ones significantly outperforming the baseline in terms of BLEU score) in all of the metrics were the simplest ones – with context sentences concatenated to the input, separated by a special token, without any changes to the model architecture. These results seem to be in line with recent development in other fields of natural language processing, where big Transformer models, trained on large datasets and many GPUs, usually outperform specialized models with an explicit problem knowledge programmed into them.

Bibliography

[1] Agrawal, R.; Turchi, M.; Negri, M.: Contextual Handling in Neural Machine Translation: Look Behind, Ahead and on Both Sides. 2018.

[2] Bahdanau, D.; Cho, K.; Bengio, Y.: Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR*. vol. abs/1409.0473. 2014. 1409.0473.
Retrieved from: http://arxiv.org/abs/1409.0473

[3] Bapna, A.; Firat, O.: Non-Parametric Adaptation for Neural Machine Translation. *CoRR*. vol. abs/1903.00058. 2019. 1903.00058.
Retrieved from: http://arxiv.org/abs/1903.00058

[4] Bar-Hillel, Y.: Demonstration of the Nonfeasibility of Fully Automatic High Quality Translation. 1960.

[5] Bawden, R.; Sennrich, R.; Birch, A.; et al.: Evaluating Discourse Phenomena in Neural Machine Translation. *CoRR*. vol. abs/1711.00513. 2017. 1711.00513.
Retrieved from: http://arxiv.org/abs/1711.00513

[6] Belinkov, Y.; Bisk, Y.: Synthetic and natural noise both break neural machine translation. *arXiv preprint arXiv:1711.02173*. 2017.

[7] Bojar, O.; Federmann, C.; Fishel, M.; et al.: Findings of the 2018 Conference on Machine Translation (WMT18). In *Proceedings of the Third Conference on Machine Translation, Volume 2: Shared Task Papers*. Belgium, Brussels: Association for Computational Linguistics. October 2018. pp. 272–307.
Retrieved from: http://www.aclweb.org/anthology/W18-6401

[8] Brown, P.; Cocke, J.; Pietra, S. D.; et al.: A statistical approach to language translation. In *Proceedings of the 12th conference on Computational linguistics-Volume 1*. Association for Computational Linguistics. 1988. pp. 71–76.

[9] Cho, K.; van Merrienboer, B.; Gülçehre, Ç.; et al.: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR*. vol. abs/1406.1078. 2014. 1406.1078.
Retrieved from: http://arxiv.org/abs/1406.1078

[10] Collobert, R.; Weston, J.; Bottou, L.; et al.: Natural language processing (almost) from scratch. *Journal of machine learning research*. vol. 12, no. Aug. 2011: pp. 2493–2537.

[11] Coughlin, D.: Correlating automated and human assessments of machine translation quality.

[12] Devlin, J.; Chang, M.-W.; Lee, K.; et al.: Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*. 2018.

[13] Eisenstein, J.: *Natural Language Processing*. MIT Press. 2018.

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

[15] García-Martínez, M.; Barrault, L.; Bougares, F.: Factored Neural Machine Translation. *CoRR*. vol. abs/1609.04621. 2016. 1609.04621.
Retrieved from: http://arxiv.org/abs/1609.04621

[16] Halliday, M. A. K.: Hasan. *Cohesion in English*. 1976.

[17] Hardmeier, C.: Discourse in statistical machine translation. a survey and a case study. *Discours. Revue de linguistique, psycholinguistique et informatique. A journal of linguistics, psycholinguistics and computational linguistics.* , no. 11. 2012.

[18] Hassan, H.; Aue, A.; Chen, C.; et al.: Achieving Human Parity on Automatic Chinese to English News Translation. *CoRR*. vol. abs/1803.05567. 2018. 1803.05567.
Retrieved from: http://arxiv.org/abs/1803.05567

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

[20] Jean, S.; Cho, K.: Context-Aware Learning for Neural Machine Translation. *CoRR*. vol. abs/1903.04715. 2019. 1903.04715.
Retrieved from: http://arxiv.org/abs/1903.04715

[21] Jean, S.; Lauly, S.; Firat, O.; et al.: Does Neural Machine Translation Benefit from Larger Context? *arXiv preprint arXiv:1704.05135*. 2017.

[22] Johnson, M.: How the statistical revolution changes (computational) linguistics. In *Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?*. Association for Computational Linguistics. 2009. pp. 3–11.

[23] Junczys-Dowmunt, M.: Dual Conditional Cross-Entropy Filtering of Noisy Parallel Corpora. *arXiv preprint arXiv:1809.00197*. 2018.

[24] Junczys-Dowmunt, M.: Microsoft's Submission to the WMT2018 News Translation Task: How I Learned to Stop Worrying and Love the Data. *CoRR*. vol. abs/1809.00196. 2018. 1809.00196.
Retrieved from: http://arxiv.org/abs/1809.00196

[25] Junczys-Dowmunt, M.; Grundkiewicz, R.; Dwojak, T.; et al.: Marian: Fast Neural Machine Translation in C++. In *Proceedings of ACL 2018, System Demonstrations*. Melbourne, Australia: Association for Computational Linguistics. July 2018. pp. 116–121.
Retrieved from: http://www.aclweb.org/anthology/P18-4020

[26] Jurafsky, D.: *Speech & language processing*. Pearson Education India. 2000.

[27] Kay, M.: Machine translation will not work. In *24th Annual Meeting of the Association for Computational Linguistics*. 1986.

[28] Kendall, G.; Wickham, G.: *Using Foucault's Methods*. Introducing Qualitative Methods series. SAGE Publications. 1999. ISBN 9780761957171.
Retrieved from: https://books.google.cz/books?id=3Zqwm4SQefoC

[29] Khayrallah, H.; Koehn, P.: On the Impact of Various Types of Noise on Neural Machine Translation. *arXiv preprint arXiv:1805.12282*. 2018.

[30] Koehn, P.: Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 conference on empirical methods in natural language processing*. 2004.

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

[32] Koehn, P.; Hoang, H.; Birch, A.; et al.: Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*. Association for Computational Linguistics. 2007. pp. 177–180.

[33] Lakew, S. M.; Cettolo, M.; Federico, M.: A Comparison of Transformer and Recurrent Neural Networks on Multilingual Neural Machine Translation. *arXiv preprint arXiv:1806.06957*. 2018.

[34] Läubli, S.; Sennrich, R.; Volk, M.: Has Machine Translation Achieved Human Parity? A Case for Document-level Evaluation. *CoRR*. vol. abs/1808.07048. 2018. 1808.07048.
Retrieved from: http://arxiv.org/abs/1808.07048

[35] Lawson, V.: Practical Experience of Machine Translation.

[36] Lei Ba, J.; Ryan Kiros, J.; E. Hinton, G.: Layer Normalization. 07 2016.

[37] Libovický, J.; Helcl, J.: Attention Strategies for Multi-Source Sequence-to-Sequence Learning. *CoRR*. vol. abs/1704.06567. 2017. 1704.06567.
Retrieved from: http://arxiv.org/abs/1704.06567

[38] Libovický, J.; Helcl, J.; Marecek, D.: Input Combination Strategies for Multi-Source Transformer Decoder. *CoRR*. vol. abs/1811.04716. 2018. 1811.04716.
Retrieved from: http://arxiv.org/abs/1811.04716

[39] Lui, M.; Baldwin, T.: langid. py: An off-the-shelf language identification tool. In *Proceedings of the ACL 2012 system demonstrations*. Association for Computational Linguistics. 2012. pp. 25–30.

[40] Manning, C. D.; Surdeanu, M.; Bauer, J.; et al.: The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*. 2014. pp. 55–60.
Retrieved from: http://www.aclweb.org/anthology/P/P14/P14-5010

[41] Maruf, S.; Haffari, G.: Document Context Neural Machine Translation with Memory Networks. *CoRR*. vol. abs/1711.03688. 2017. 1711.03688.
Retrieved from: http://arxiv.org/abs/1711.03688

[42] Maruf, S.; Martins, A. F. T.; Haffari, G.: Selective Attention for Context-aware Neural Machine Translation. *CoRR*. vol. abs/1903.08788. 2019. 1903.08788.
Retrieved from: http://arxiv.org/abs/1903.08788

[43] Mey, J.: *Pragmatics: An Introduction*. Wiley. 2001. ISBN 9780631211327.
Retrieved from: https://books.google.cz/books?id=TT_TF4sM6lcC

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

[45] Moussallem, D.; Arcan, M.; Ngomo, A. N.; et al.: Augmenting Neural Machine Translation with Knowledge Graphs. *CoRR*. vol. abs/1902.08816. 2019. 1902.08816.
Retrieved from: http://arxiv.org/abs/1902.08816

[46] Ott, M.; Edunov, S.; Grangier, D.; et al.: Scaling Neural Machine Translation. *CoRR*. vol. abs/1806.00187. 2018. 1806.00187.
Retrieved from: http://arxiv.org/abs/1806.00187

[47] Papineni, K.; Roukos, S.; Ward, T.; et al.: BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002. pp. 311–318.

[48] Pfülb, B.; Gepperth, A.; Abdullah, S.; et al.: Catastrophic forgetting: still a problem for DNNS. In *International Conference on Artificial Neural Networks*. Springer. 2018. pp. 487–497.

[49] Popel, M.; Bojar, O.: Training Tips for the Transformer Model. *CoRR*. vol. abs/1804.00247. 2018. 1804.00247.
Retrieved from: http://arxiv.org/abs/1804.00247

[50] Popović, M.: chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*. 2015. pp. 392–395.

[51] Post, M.: A Call for Clarity in Reporting BLEU Scores. *CoRR*. vol. abs/1804.08771. 2018. 1804.08771.
Retrieved from: http://arxiv.org/abs/1804.08771

[52] Radford, A.; Wu, J.; Child, R.; et al.: Language models are unsupervised multitask learners.

[53] Schmied, J.; Haase, C.; Povolná, R.: *Complexity and Coherence: Approaches to Linguistic Research and Language Teaching*. REAL studies. Cuvillier Verlag. 2007. ISBN 9783867272155.
Retrieved from: https://books.google.cz/books?id=XiMLAQAAMAAJ

[54] Sennrich, R.; Birch, A.; Currey, A.; et al.: The university of edinburgh's neural MT systems for WMT17. *arXiv preprint arXiv:1708.00726*. 2017.

[55] Sennrich, R.; Firat, O.; Cho, K.; et al.: Nematus: a Toolkit for Neural Machine Translation. In *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics*. Valencia, Spain: Association for Computational Linguistics. April 2017. pp. 65–68.
Retrieved from: http://aclweb.org/anthology/E17-3017

[56] Sennrich, R.; Haddow, B.: Linguistic Input Features Improve Neural Machine Translation. *CoRR*. vol. abs/1606.02892. 2016. 1606.02892.
Retrieved from: http://arxiv.org/abs/1606.02892

[57] Sennrich, R.; Haddow, B.; Birch, A.: Improving Neural Machine Translation Models with Monolingual Data. *CoRR*. vol. abs/1511.06709. 2015. 1511.06709.
Retrieved from: http://arxiv.org/abs/1511.06709

[58] Sennrich, R.; Haddow, B.; Birch, A.: Neural Machine Translation of Rare Words with Subword Units. *CoRR*. vol. abs/1508.07909. 2015. 1508.07909.
Retrieved from: http://arxiv.org/abs/1508.07909

[59] Serrà, J.; Surís, D.; Miron, M.; et al.: Overcoming catastrophic forgetting with hard attention to the task. *CoRR*. vol. abs/1801.01423. 2018. 1801.01423.
Retrieved from: http://arxiv.org/abs/1801.01423

[60] Shterionovα, D.; Casanellasβ, P. N. L.; Superboβ, R.; et al.: Empirical evaluation of NMT and PBSMT quality for large-scale translation production. In *Conference Booklet*. page 74.

[61] Srivastava, N.; Hinton, G.; Krizhevsky, A.; et al.: Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*. vol. 15, no. 1. 2014: pp. 1929–1958.

[62] Sukhbaatar, S.; Grave, E.; Bojanowski, P.; et al.: Adaptive Attention Span in Transformers. 2019.

[63] Sutskever, I.; Vinyals, O.; Le, Q. V.: Sequence to Sequence Learning with Neural Networks. *CoRR*. vol. abs/1409.3215. 2014. 1409.3215.
Retrieved from: http://arxiv.org/abs/1409.3215

[64] Szegedy, C.; Vanhoucke, V.; Ioffe, S.; et al.: Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016. pp. 2818–2826.

[65] Toral, A.; Castilho, S.; Hu, K.; et al.: Attaining the unattainable? Reassessing claims of human parity in neural machine translation. *arXiv preprint arXiv:1808.10432*. 2018.

[66] Tu, Z.; Liu, Y.; Lu, Z.; et al.: Context Gates for Neural Machine Translation. *CoRR*. vol. abs/1608.06043. 2016. 1608.06043.
Retrieved from: http://arxiv.org/abs/1608.06043

[67] Vanmassenhove, E.; Hardmeier, C.; Way, A.: Getting gender right in neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018. pp. 3003–3008.

[68] Vaswani, A.; Shazeer, N.; Parmar, N.; et al.: Attention Is All You Need. *CoRR*. vol. abs/1706.03762. 2017. 1706.03762.
Retrieved from: http://arxiv.org/abs/1706.03762

[69] Voita, E.; Serdyukov, P.; Sennrich, R.; et al.: Context-Aware Neural Machine Translation Learns Anaphora Resolution. *CoRR*. vol. abs/1805.10163. 2018. 1805.10163.
Retrieved from: http://arxiv.org/abs/1805.10163

[70] Wang, L.; Tu, Z.; Way, A.; et al.: Exploiting Cross-Sentence Context for Neural Machine Translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017.

[71] Wen, S.; Itti, L.: Overcoming catastrophic forgetting problem by weight consolidation and long-term memory. *CoRR*. vol. abs/1805.07441. 2018. 1805.07441.
Retrieved from: http://arxiv.org/abs/1805.07441

[72] Wu, Y.; Schuster, M.; Chen, Z.; et al.: Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR*. vol. abs/1609.08144. 2016. 1609.08144.
Retrieved from: http://arxiv.org/abs/1609.08144

[73] Zhang, J.; Luan, H.; Sun, M.; et al.: Improving the Transformer Translation Model with Document-Level Context. *arXiv preprint arXiv:1810.03581*. 2018.