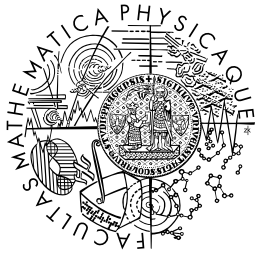


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Thuong-Hai Pham

Exploiting Sentence Structure in Neural Machine Translation

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	RNDr. Ondřej Bojar, PhD
Study programme:	Informatics
Study branch:	Computational Linguistics

Prague 2018

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

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

Prague, 20.07.2018 signature of the author

First, I would like to express my gratitude to RNDr. Ondřej Bojar, Ph.D, my supervisor, for his dedicated guidance.

I deeply appreciate the financial support from the Erasmus Mundus scholarship.

I also want to thank doc. RNDr. Markéta Lopatková, Ph.D, Mrs. Bobbye Pernice and Mr. Mike Rosner for always helping me with the important matters before and during my Erasmus Mundus LCT program.

I am very grateful to Martin Popel for his recommendations regarding the technical aspects of this thesis, and XiaoYu Bai for her proofreading.

I would like to give a heartfelt thank to my family for always being supportive of my choices in life.

And thanks to you whoever is reading, for your interest.

Title: Exploiting Sentence Structure in Neural Machine Translation

Author: Thuong-Hai Pham

Institute: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Ondřej Bojar, PhD, Institute of Formal and Applied Linguistics

Abstract: Neural machine translation has been lately established as the new state of the art in machine translation, especially with the Transformer model. This model emphasized the importance of self-attention mechanism and suggested that it could capture some linguistic phenomena. However, this claim has not been examined thoroughly, so we propose two main groups of methods to examine the relation between these two. Our methods aim to improve the translation performance by directly manipulating the self-attention layer. The first group focuses on enriching the encoder with source-side syntax with tree-related position embeddings or our novel specialized attention heads. The second group is a joint translation and parsing model leveraging self-attention weight for the parsing task. It is clear from the results that enriching the Transformer with sentence structure can help. More importantly, the Transformer model is in fact able to capture this type of linguistic information with guidance in the context of multi-task learning at nearly no increase in training costs.

Keywords: attention machine translation dependency neural network

Contents

Introduction

Neural machine translation (NMT) has been lately established as the new state of the art in machine translation (MT). To achieve this result, most NMT models highly rely on the attention mechanism [Bahdanau et al., 2015].

The dependence of NMT models on the attention mechanism is further strengthened in the Transformer model [Vaswani et al., 2017]. This model gets rid of the recurrent neural network (RNN), and replaces it with self-attention layers. It was claimed that this self-attention mechanism can capture some linguistic structures and phenomena.

Although Belinkov et al. [2017] and Shi et al. [2016] analyzed the amount of linguistic information (POS tags and syntax, respectively) NMT systems could capture, their analyses only applied to systems without attention mechanism. More notably, Ghader and Monz [2017] examined the similarities and differences between attention and alignment matrix. However, their goal was merely to inspect the alignment, not the linguistic knowledge of the source sentence nor target sentence. More importantly, most of the previous works attempting to examine the link between NMT and linguistic information focused on the common but outdated sequence-to-sequence model [Sutskever et al., 2014].

This thesis aims to fill the gap and examine the relation between linguistic structures of source sentences, e.g. dependency syntax [Mel'čuk, 1988], and the self-attention mechanism (within the sentence) in the relatively new Transformer's encoder, by focusing on two basic questions:

- Does explicitly feeding source-side syntactic knowledge to an NMT system help translation quality?
- If not, is the attention mechanism perhaps already capturing this information?

Outline

Apart from this introduction, the thesis is organized into six chapters:

Chapter 1 briefly introduces the theoretical background regarding machine translation and several forms of linguistic information.

Chapter 2 reviews the previous related works that our methods are built upon.

Chapter 3 presents our first set of attempts to improve the Transformer's model by enriching the encoder's self-attention with information about the source sentence structure.

Chapter 4 proposes our joint translating and parsing model by promoting the interpretation of the self-attention layers in the encoder.

Chapter 5 presents the datasets that are used to evaluate our approaches and details about our experimental systems.

Chapter 6 reports the results of our experiments and further analyzes the training time and the behavior of the neural network.

1. Theoretical Background

This chapter presents the basic theories and knowledge that this thesis is built upon.

1.1 Machine Translation

Machine translation is an area of computational linguistics aiming to automatically translate text or speech from the *source* language to the *target* language.

Machine translation is distinct from computer-aided translation in which human translators use computer software to support the translation process. Machine translation should be performed on a fully automatic basis without any human interaction. Obviously, a machine translation system can be used during computer-aided translation.

The two most common types of MT system are rule-based and statistical. These types differ in the way that they are built, but are often combined within the same system which is then called *hybrid MT*.

Rule-based machine translation (RBMT) considers language as a set of rules capturing all the relevant regularities in morphology, syntax, or other layers of language description. In order to translate a sentence, it employs a predefined set of rules within the language to break down the sentence. Then, another set of rules links between the source and target language, and a robust bilingual dictionary is used to carry out the translation. These rules are essential for building such a system, but they are usually expensive to construct and must be created anew for each new language pair.

On the other hand, *statistical machine translation (SMT)* does not use linguistic rules to translate but instead a statistical model. For each unit of the source sentence, there are several possible translations associated with a probability. The goal of the SMT system is to select these translations for each source unit such that they cover the source sentence with the highest translation probability. The statistical model helps this selection process, parameters of which must be derived from a sufficient amount of data. The data in this case is called the *bilingual corpus*, a set of pairs, each consisting of a sentence from the source language and the corresponding sentence in the target language.

From the mid 2010s, the field of MT has seen a new group of systems emerge and then outperform the two previously mentioned types to establish the new state of the art. These MT systems are actually not too different from SMT as they are also statistical learning models. However, they employ deep neural networks, which have been very successful in computer vision and simpler linguistic tasks, to model the translation task. Hence, this type is referred to with a distinct name, *neural machine translation*, which will be discussed in detail in the following section.

1.1.1 Neural Machine Translation

As mentioned above, a *neural machine translation* system uses a deep neural network to learn the statistical model for machine translation. Unlike the statistical

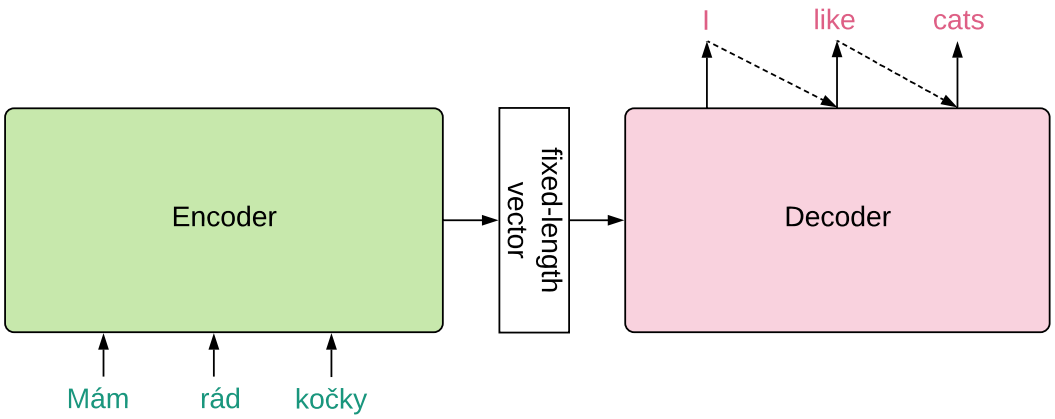


Figure 1.1: `seq2seq` model.

MT, where all types (word-based, phrase-based or syntax-based) consist of several sub-components that need to be trained separately, NMT can be built and trained in a single neural network. Such a system is referred to as end-to-end, in which the model is trained at once, directly mapping an input source sentence to its associated sentence in the target language.

The use of neural networks in MT dates back to the works of Castaño et al. [1997] and Neco and Forcada [1997]. After a break, NMT returned and has started to show promising results with the sequence-to-sequence (`seq2seq`) model proposed by Sutskever et al. [2014]. The idea was based on the assumption that when translating a sentence (in text or speech), a human reads or listens to the whole sentence, understands it, then starts to write down the translated sentence token by token. Therefore, `seq2seq` consists of two main components: an *encoder* and a *decoder* (Figure 1.1). The encoder consumes the source sentence and produces a fixed-length vector encoding the "meaning" of the sentence. This vector is then fed to the decoder. Finally, the decoder produces the target sentence in the target language token-by-token, based on the context vector and the previously translated tokens. In the `seq2seq` model, the encoder and decoder are both modelled by a recurrent neural network, or its variant, *long-short term memory* (LSTM) [Hochreiter and Schmidhuber, 1995].

1.1.2 Attention Mechanism

The assumption that human translators keep in mind the meaning of the entire source sentence is arguable. Some might argue that the translator has to look back to some part of the source sentence during the translation process.

We are yet to know which is the correct way our brain works during the translation process. However, from the technical point of view, the limitation of the encoder-decoder approach is that the encoder has to output a fixed-length vector from the source sentence, which is then used in the decoder to output the target sentence. This means that every decoded token w is condition from the same vector, which captures the whole source sentence, instead of from some parts, e.g. clauses, phrases, etc., of the source sentence which are most associated with w. In addition, compressing all information into a fixed-length vector leads

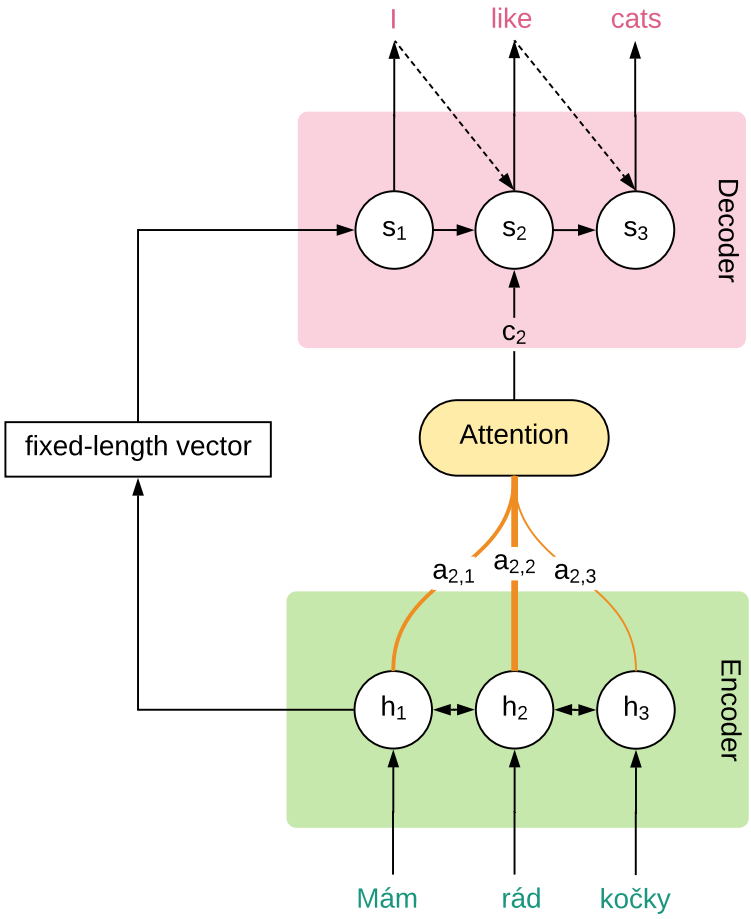


Figure 1.2: `seq2seq` model with Bahdanau attention.

to the loss of information when the sentence becomes longer. This is where the attention mechanism is able to help `seq2seq`.

Figure 1.2 illustrates this mechanism. When translating the second token, the model does not only take into account the information from s_1, but also looks back to the encoder. The main computational steps of the attention layer are:

- Compute the *attention weights* $a_{2,1}$, $a_{2,2}$, $a_{2,3}$.
- Compute the *context vector* $c_2 = a_{2,1}h_1 + a_{2,2}h_2 + a_{2,3}h_3$.
- Feed c_2 to the computation of s_2.

Bahdanau et al. [2015] proposed the formulas to compute the attention weight $a_{i,j}$ as follows:

1. $e_{i,j} = att(s_{i-1}, h_j)$, where $e_{i,j}$ is the *attention energy*, s_{i-1} is the previous hidden state of the decoder, h_j is the hidden state at position j in the encoder.
2. $a_{i,j} = \frac{exp(e_{i,j})}{\sum_t exp(e_{i,t})}$.

The function *att* is simply a dot product between two vectors. The second step is the *sigmoid function*, ensuring that $\sum_t a_{i,t} = 1$, i.e. a distribution.

This attention mechanism brings a significant improvement to the `seq2seq` model. In some examples, the attention weight matrix is visually similar to the alignment matrix from SMT. Hence, the attention was also considered as soft-alignment because it does not need to strictly match tokens between the two sentences.

Luong et al. [2015] called this type of attention *global attention*, to differentiate from their proposed *local attention*. In local attention, the attention layer selects one token from the source sentence, then distributes the attention weights, based on a Gaussian distribution, to all neighbors of that token within a fixed-size window. Due to the considerations of space, we will not elaborate on the local attention, which was discussed by the authors as "complicated to implement and train". (see the discussion[1] for details).

1.2 Evaluation

In this section, we discuss several methods to evaluate MT systems, which are divided into two types: manual evaluation methods and automatic metrics.

1.2.1 Manual Evaluation

Evaluating MT systems has remained one of the most important problems in the field. Hence, there are many proposed methods, even on how humans should judge the performance of machine-translated text. Most of the methods simply judge the hypotheses produced by MT systems, treating the systems as black boxes.

A very straightforward evaluation is to directly assess the adequacy and fluency of whole sentences [Graham et al., 2013]. By adequacy, the annotator is expressing to what extent MT captures the meaning of the reference sentence. Fluency, on the other hand assesses the grammatical correctness and overall quality of the target sentence regardless the source. In Graham, fluency is merely used to break ties in adequacy. Another option is to rank full sentences or constituents, i.e. parts of sentences, from several MT systems. Yet Bojar et al. [2011] showed that it is problematic to interpret this manual ranking.

As an alternative, task-based methods evaluate if information from MT output is as useful as the original sentence. Nevertheless, this usefulness is also a vague concept and it is hard to measure.

Although several manual evaluation methods introduced various strategies to overcome the subjectiveness of human judges, this type of evaluation is generally expensive in terms of both time and money.

1.2.2 Automatic Metrics

Given the problems of manual evaluation methods described above, it is natural to try to find a fast, cheap, deterministic and replicable metric. Moreover, it would be a plus if the metric allowed automatic model optimization.

[1] `https://github.com/tensorflow/tensorflow/issues/10842#issuecomment-372546360`

With these properties, the proposed metric can be used to check progress, allow researchers to iterate and evaluate their proposals faster and speed up the development of the field.

The *BLEU* metric (Bilingual Evaluation Understudy, Papineni et al. [2002]) is one of these automatic evaluation metrics, which is widely used in the field of MT. It evaluates an output (sentence or corpus) of an MT system (the candidate) by comparing it with correct translations (the references).

The two main components of BLEU are the n-grams precisions and length of the candidate. Precision is very commonly used in the machine learning field. In the case of BLEU, it measures the percentage of correct n-grams in the candidate. The trivial case is unigram ($n = 1$) precision which is merely the ratio of the number of tokens shared between candidate and reference divided by the number of tokens in the candidate. However, this simple definition of precision would not be very precise in some cases, for example:

Candidate: that that that
Reference: I think that it is not that bad

The straightforward (lowercase) unigram precision of the above example is 1.0 (100%), even though only two *that* unigrams in the candidate are matched with the two unigrams in the reference. That is to say, the number of n-grams shared between the candidate and the reference should be clipped to the number of n-grams that appear in the reference. Having that modification, the *modified n-gram precision* in BLEU is computed as follows:

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n-gram \in C} Count_{clip}(n-gram)}{\sum_{C' \in \{Candidates\}} \sum_{n-gram' \in C'} Count(n-gram')} \tag{1.1}$$

The second problem BLEU has to deal with is erroneously short candidates. Take the following example:

Candidate: that
Reference: I think that it is not that bad

Although the candidate definitely does not express enough information compared to the reference, the precision of this case is 1.0. To penalize such output from MT systems, BLEU introduced the *brevity penalty* where c and r are the length of the candidate and the length of the reference, respectively.

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{1.2}$$

When there are more than one reference, r is called the *effective reference length* and it is taken as the length of the reference that is closest to the length of the candidate. It is important to note that which reference is the closest varies between implementations of BLEU, see the example below. Both references' lengths are one token different away from the candidate.

Candidate: I like
Reference 1: I like it
Reference 2: I

We advise the reader to use the official BLEU evaluation script used by the Workshop of Machine Translation (WMT) shared task,[2] or its Python reimplementation.[3]

Combining those two main components, the BLEU score is defined as follows:

$$BLEU = BP \cdot exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{1.3}$$

Specifically, BLEU computes the n-grams precisions p_n of the given candidate and references (by default from unigrams to 4-grams). It then geometrically averages them with predefined weights w_n (all set to 1/4 by default), and scales down the score in the case of inadequately short candidates with the brevity penalty.

Experimental results showed that BLEU is highly correlated with human evaluators if several reference translations are used and the BLEU scores are sufficiently high [Bojar et al., 2010]. However, BLEU is overly sensitive to word forms and sequences of tokens. There are several proposals to mitigate this, such as using:

- Lemmas or deep-lemmas instead of word forms as in SemPOS [Kos and Bojar, 2009].
- Sequences of characters, e.g. chrF3 [Popovic, 2015] which is f-score of character 6-grams.
- Shorter and gappy sequences, e.g. BEER [Stanojević and Sima'an, 2014] uses characters and also pairs of (not necessarily adjacent) words.

1.3 Linguistic Information

In this section, we would like to introduce two types of linguistic information that our proposed methods exploit to enhance the translation performance. Section 1.3.1 discusses the concept of dependency grammar, while Section 1.3.2 reviews various POS tagging systems across languages.

1.3.1 Dependency Grammar

The so-called *dependency grammar* is in fact not a single consistently established grammar but a wide range of variants that share several basic assumptions. The primary underlying idea is a syntactic structure which consists of lexical items, connected by binary asymmetric relations. These relations are called dependencies. Although it is said that this concept has been used early in Panini's work for Sanskrit grammar around the 6$^{\text{th}}$ to 4$^{\text{th}}$ century BCE, its systematic introduction is due to Tesnière [1959]. The two nodes involved in this type of relation are called *head* (*governor*) and *dependent* (*subordinate*).

In order to visualize these relations in a sentence, a tree-like structure named *dependency tree* is used. Aside from dependency tree, another common tree-like

[2] `ftp://jaguar.ncsl.nist.gov/mt/resources/mteval-v13a.pl`

[3] `https://github.com/mjpost/sacreBLEU`

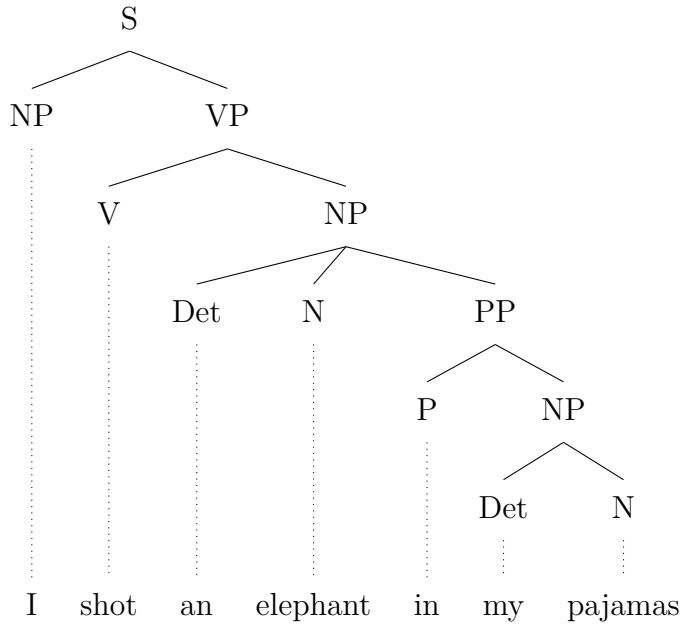


Figure 1.3: Phrase-structure grammar tree.

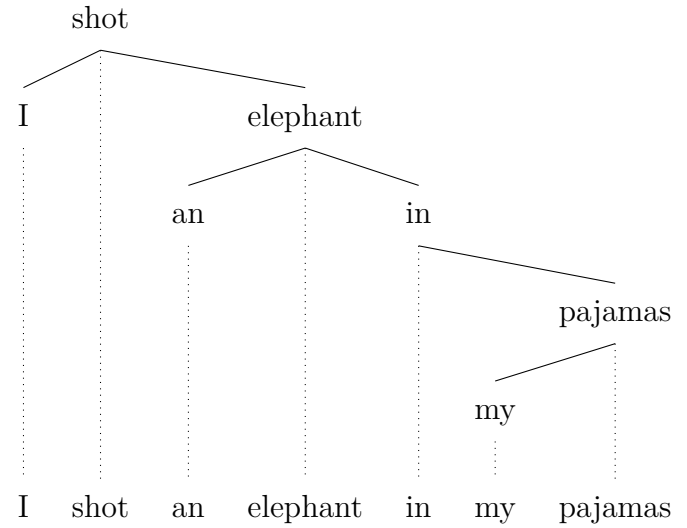


Figure 1.4: Dependency grammar tree.

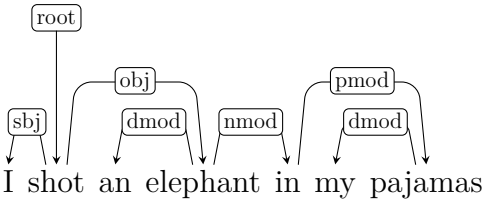


Figure 1.5: Dependency grammar tree with arc labels.

structure used in describing the grammar of the sentence is the phrase-structure tree.

Figures 1.3 and 1.4 present an example of a phrase-structure and a dependency tree, respectively. While the phrase-structure tree represents phrases (as non-terminal nodes) with their structural categories, the dependency tree depicts the head-dependent relations (with directed arcs) between its lexical items. Figure 1.5 also introduces a dependency tree with arc labels, known as *dependency labels*, which specify the functional relation that holds between each pair of lexical items.

In the examples, the noun *elephant* is the dependent of the verb *shot* and the nature of the dependency relation is specified by the label, which indicates that *elephant* is the object of *shot* (Figure 1.5). The phrase-structure tree has a slightly different analysis. On the 4^{th} level of the tree (Figure 1.3), the prepositional phrase (PP) is comprised of a preposition (P) and a noun phrase (NP), spanning over the phrase *"in my pajamas"* of the sentence.

It should also be pointed out that the convention which two nodes are related, which of them is marked as the dependent and which as the head, and what is the dependency label sometimes varies between treebanks. For example, the Universal Dependencies 2.0 (UD 2.0, Nivre et al. [2017]) has a different set of labels and sometimes different ways to attach a certain token to its head, than the Prague Dependency Treebanks (PDT, Hajič et al. [2006]).

Dependency Parsing

Dependency parsing is a syntactic analysis of lexical items focused on finding the dependency relation between tokens in a sentence. In other words, dependency parsing is the process that takes a sentence as the input and outputs the dependency tree discussed in the previous section. The dependency parser can be built using a set of predefined rules or by learning from data. The dataset that a parser learns from is called a *treebank*. Data-driven dependency parsing has two prominent approaches: *transition-based* or *graph-based parsing*.

The transition-based parsing maintains a so-called *configuration*. Then, it uses an *oracle* to decide an action, e.g. whether to attach a token to another token, or to change the current configuration. A configuration in a stack-based shift-reduce parser includes:

- A *buffer* containing tokens from the input sentence which have yet to be processed.
- A *stack* containing all tokens that are being processed.
- A *set of dependency relations* where each member is an edge in the final parse tree.

The oracle of a transition-based parser is actually a classifier, which can be built with any data-driven approach that can solve a classification problem, for example a neural network [Chen and Manning, 2014]. The input of this classifier includes several features extracted from the configuration, while the output is an action from a set of possible actions. Chen and Manning proposed a rich set of features including word form, POS tags and currently known arc labels of the top 3 tokens in the stack and the buffer, and also selected children of these tokens. The main and only component of a transition-based parser that needs to be trained is the oracle.

Compared to transition-based parsing, graph-based parsing is a more direct approach. Basically, it transforms the problem of dependency parsing to finding the maximum spanning tree in a connected graph. First of all, a graph-based parser considers all tokens in the sentence as a complete graph. Utilizing the linguistic information of the tokens, e.g. word forms, lemmas, part-of-speech tags (Section 1.3.2), etc., the parser predicts the weight of each possible directed edge (u, v), which indicates the likelihood that token u is the head of token v. The parser then finds the maximum spanning tree from the graph based on the computed probabilities. This maximum spanning tree is the optimal parse tree of the input sentence.

The task of assigning a dependency label to each dependency relation (*dependency tagging*) may be performed during the parsing process or after having obtained the dependency tree. Nevertheless, we will not elaborate on this task, which is of limited relevance to the present thesis.

1.3.2 Part-of-Speech Tagging

By definition, a *part of speech (POS)* is a category of words that have similar grammatical properties. If two words are assigned the same POS tag, they have similar grammatical functions in the structure of sentences.

It is important to note that the set of POS tags varies among languages. For example, Vietnamese has the tag of nominal classifiers, which English does not. For English, the POS tags from the Penn treebank project [Marcus et al., 1993] are commonly used (Table 1.1).

On the other hand, this small POS tag set is arguably not sufficient for morphologically rich languages such as Czech. In the Czech language, a complex system of morphological tags is used. Each morphological tag is an encoded sequence. This sequence consists of 15 characters whose functions are described in Table 1.2.

The very first position denotes 10 basic POS tags (Table 1.3), while the second position adds more details about the POS tags. Hence, the first two characters of a morphological tag can be considered equivalent to the POS tag in the Penn treebank.

For example, the morphological tag of *rezignoval* in the sentence "*Myslíš, že tě požádají, abys rezignoval?*" is `VpYS---XR-AA---`, which means:

- V: verb.
- p: verb, past participle, active.
- Y: masculine.

- S: singular.
- X: any person.
- R: past tense.
- A: affirmative (not negated).
- A: active voice.
- The hyphen (-) indicates that information at that position is not applicable.

No.	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential there
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	to
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Table 1.1: List of POS tags used in the Penn treebank project.

No.	Name	Description
1	POS	Part of Speech
2	SUBPOS	Detailed Part of Speech
3	GENDER	Gender
4	NUMBER	Number
5	CASE	Case
6	POSSGENDER	Possessor's Gender
7	POSSNUMBER	Possessor's Number
8	PERSON	Person
9	TENSE	Tense
10	GRADE	Degree of comparison
11	NEGATION	Negation
12	VOICE	Voice
13	RESERVE1	Unused
14	RESERVE2	Unused
15	VAR	Variant, Style, Register, Special Usage

Table 1.2: 15 positions of a morphological tag in the Czech language.

Value	Description
A	Adjective
C	Numeral
D	Adverb
I	Interjection
J	Conjunction
N	Noun
P	Pronoun
V	Verb
R	Preposition
T	Particle
X	Unknown, Not Determined, Unclassifiable
Z	Punctuation (also used for sentence boundary and token)

Table 1.3: Possible values in the first position of a morphological tag in the Czech language.

2. Literature Review

Recent attempts to incorporate linguistic information, especially syntactic information, to NMT systems can be roughly classified into two main categories: enriching the encoder or decoder and multi-task learning.

This chapter is divided into three sections: the first two Sections 2.1 and 2.2 discuss the two main categories mentioned above, and Section 2.3 is dedicated to a more detailed review of our baseline, the Transformer model.

2.1 Enriching the Encoder or Decoder

The main motivation of these methods is to make syntactic information known to the NMT system with the expectation that it might help to improve translation quality.

A very straightforward method to enrich the encoder with syntax is to input this type of information alongside with source words embeddings. The `seq2seq` model with Bahdanau's attention takes word embeddings as input. Sennrich and Haddow [2016] reused this model but replaced the input with the concatenation of word embeddings and linguistic features including lemmas, subword tags, POS tags and dependency labels. The authors reported a significant improvement over the baseline with this simple approach.

Also in this direction, Eriguchi et al. [2016] combined a sequence-based encoder with a tree-based encoder. This tree-based encoder is a modified version of Tree-LSTM [Tai et al., 2015]. In Tree-LSTM, the hidden state vectors are not passed from left to right but from children to parents in a tree (Figure 2.1). The tree structure must be fed in together with the sentence. Hence, the encoder is explicitly aware of the syntactic tree and learns the context vectors from it.

In contrast, Chen et al. [2017] kept the sequence-based LSTM intact. They replaced the global attention (Section 1.1.2) with their syntax-directed attention. The proposed attention is analogous to Luong's local attention. The only difference is how the distance between two words in the source sentence is computed. In Chen's syntax-directed attention, the distance is the minimum number of edges to be traversed to reach one node from another in the dependency tree, in Luong's, it is the sequential distance.

In a simpler way, Li et al. [2017] proposed to incorporate a sequence of structural label (POS tags) to the encoder's attention by feeding these tags to an RNN. While not actually focusing on enhancing the encoder, Cohn et al. [2016] introduced structural biases to the encoder-decoder attention function. These biases include the relative position between the target and source tokens, fertility, Markov condition and bilingual symmetry. Using this approach, researchers enabled the decoder to attend better to hidden states of the encoder. It was reported that the proposed approach brought consistent improvements compared to the attentional model (`seq2seq` with Bahdanau's attention).

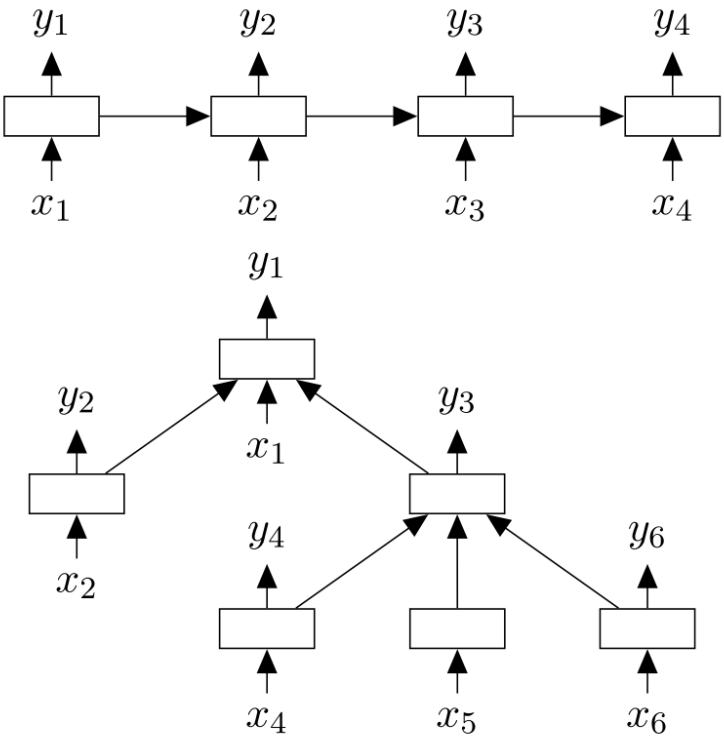


Figure 2.1: Standard LSTM (top) and tree-LSTM (bottom) (adapted from Tai et al. [2015])

2.2 Multi-Tasking

By training the model to parse and translate simultaneously, it is expected that one task can be improved using the knowledge induced from the other task. Eriguchi et al. [2017] combined the translation and dependency parsing tasks by sharing the translation encoder hidden states with the buffer hidden states in a shift-reduce parsing model Dyer et al. [2016].

While aiming at the same goal, Aharoni and Goldberg [2017] proposed a very simple method. Instead of modifying the model structure, they represented the target sentence as a linearized, lexicalized constituency tree. Subsequently, a `seq2seq` model was used to translate the source sentence to this linearized tree, i.e. indeed performing the two tasks. Le et al. [2017] made use of the same trick on dependency trees. Unfortunately, their linearization algorithm was not able to traverse a non-projective tree.

The evidence presented in these papers suggests that there is improvement on the BLEU score [Papineni et al., 2002]. The performance in the secondary task, i.e. parsing, is however not reported. Going in the opposite direction, Shi et al. [2016] have done an in-depth analysis to further examine the usefulness of NMT model for the purposes of syntactic parsing. Their work pointed out which types of syntactic relations/labels were better predicted by the `seq2seq` MT model.

Aiming to improve both tasks, Tran and Bisk [2018] used two different components in the encoder, one to produce the content and the other to produce the dependency matrix. While the content is the output of the standard bidirectional LSTM as in the vanilla `seq2seq` model, the dependency matrix is produced by

a head word selection layer, which is modelled by self-attention (Section 2.3.3). Then both are fed to the decoder using a modified encoder-decoder attentional layer. In fact, this model is enriching the decoder while still learning to do multi-tasking.

The models of multi-tasking described so far use a new network component to produce the parse. Hence, this "built-in" parser is able to make use of several shared layers with the NMT model. We would argue that this commonly used setting could hardly answer our research question number 2, namely whether the encoder is able to capture the source syntax. The reason is that the parser itself is a neural network or another sufficiently complex classifier so that it can induce syntax from its inputs on its own. In other words, a good performance of these multi-task NMT systems in parsing confirms only that the information available to the parsing component preserves sufficient details to create the parse, not that the NMT actually employs syntactic information. Therefore, one cannot answer our second research question even if the joint model performs perfectly in parsing.

In summary, the evidence presented in the literature suggests that various methods to feed syntactic information to NMT helped translation. However, little is known about whether or not NMT already captures syntactic information within the model itself, which is one of the two main questions we attempt to answer with our proposals in the following sections.

2.3 The Transformer Model

Before starting to discuss our main proposals in the next chapter. We would like to briefly introduce the reader to a novel NMT model that has established the state-of-the-art results in translation — the Transformer model [Vaswani et al., 2017]. This model has three interesting features which all our proposed methods exploit and are built upon: the self-attention mechanism, multi-head attention and positional encoding. We now go over each of them in separate sections.

2.3.1 Self-Attention

The previous studies, reviewed in Sections 2.1 and 2.2, have examined mostly the `seq2seq` model with RNNs (LSTMs) as the backbone, whose components are hard to exploit and modify. The purpose of (bidirectional) LSTMs in the encoder and decoder is to learn the representation of one token using information from its neighbors. The Transformer model eliminates the need for LSTMs in both encoder and decoder and replaces them with self-attention layers. These self-attention layers were introduced by Cheng et al. [2016], under the term *intra-attention*. They have been used in natural language inference, sentiment analysis and language modelling. The *intra-* or *self-* prefixes suggest that this attention mechanism works within the input sentence, instead of between encoder and decoder as in Section 1.1.2. Although the procedures to compute these two types of attention are similar, the attention layer in Transformer has to perform an additional projection step.

Figure 2.2 presents an example of one attention layer in the Transformer. The attention in Transformer builds upon the notion of *keys*, *values* and a *query* defined by the following steps:

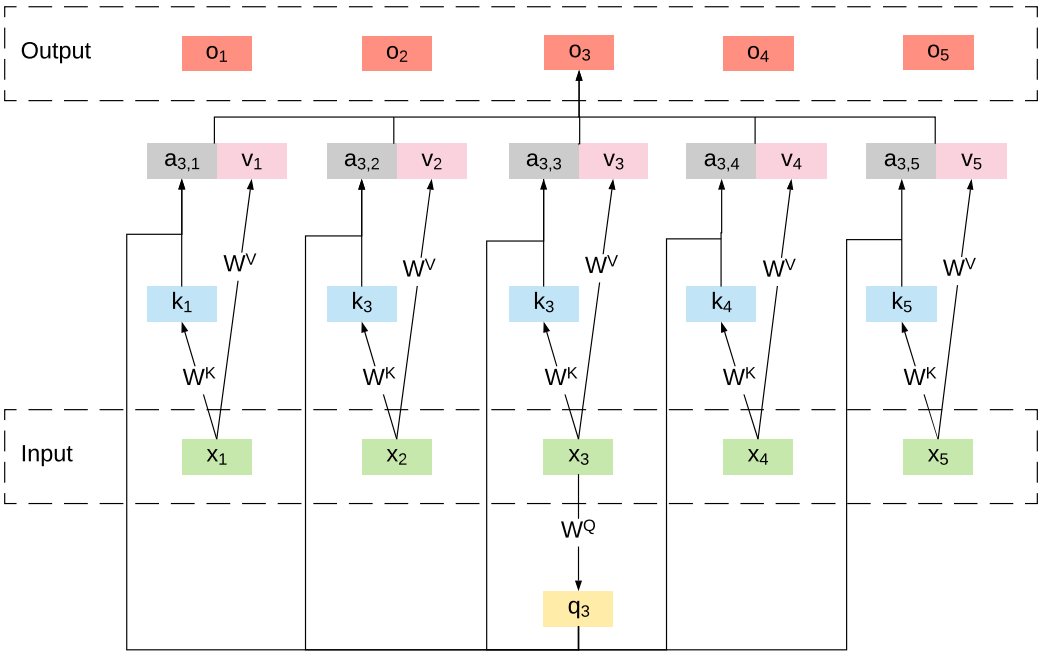


Figure 2.2: A self-attention layer in the Transformer, computing the output o_3.

1. Project the input x_i to key k_i, value v_i, and query q_i using projection matrices W^K, W^V, W^Q, respectively.

2. Compute the attention energy $e_{i,j}$ as the dot product of query q_i and key k_j, divided by the square root of the key's dimension d_k.

3. Compute the attention weight $a_{i,j} = \frac{exp(e_{i,j})}{\sum_t exp(e_{i,t})}$.

4. Compute the output $o_i = \sum_j a_{i,j} v_j$, i.e. mix the values using the attention weights.

Figure 2.3 illustrates an example of attention weight in a self-attention layer.

One significant advantage of self-attention over RNN is that the network can be processed in parallel. Parallelization within the layer in RNN is impossible because of the sequential nature of this structure.

2.3.2 Multi-Head Attention

The paper further refined the attention layer by concatenating several attention layers into one "multi-head attention" layer. In Figure 2.4, h independent attention layers are trained in parallel. The outputs of all h attentions are concatenated, then projected through a linear layer.

This multi-head attention enhances the ability of attention mechanism in two ways:

- It makes it possible for the model to attend to various types of information at different positions. Although the attention is able to focus on several

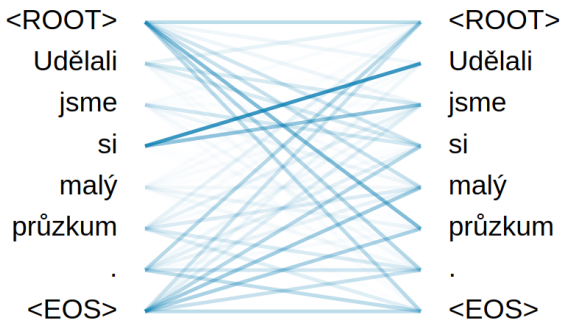


Figure 2.3: Self-attention example. The shade of lines denotes attention weight between the layer's output (right) and layer's input (left).

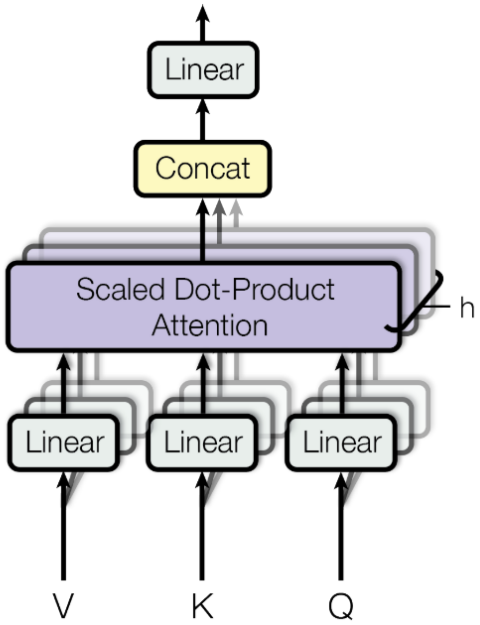


Figure 2.4: Multi-head attention layer (adapted from Vaswani et al. [2017]).

positions at one layer, multi-head attention can enable the model to attend to different patterns, and various positions in each patterns. For example, a noun in a sentence can attend to all related adjectives in the first attention head. In addition, it can also attend to the pronouns referring to it in the second head.

- The multi-head attention also allows the keys, queries and values to come from multiple representation subspaces, because of different projection matrices W^K, W^Q, W^V in each head.

2.3.3 Positional Encoding

One prominent problem of replacing LSTMs with self-attention layers is that the model has no notion of token positions. With the sequential information passing in bidirectional LSTMs, each token is aware of which tokens are before and which are after it. The self-attention layer simply computes weights using the dot product of tokens' vectors, hence it considers the input as a bag of words.

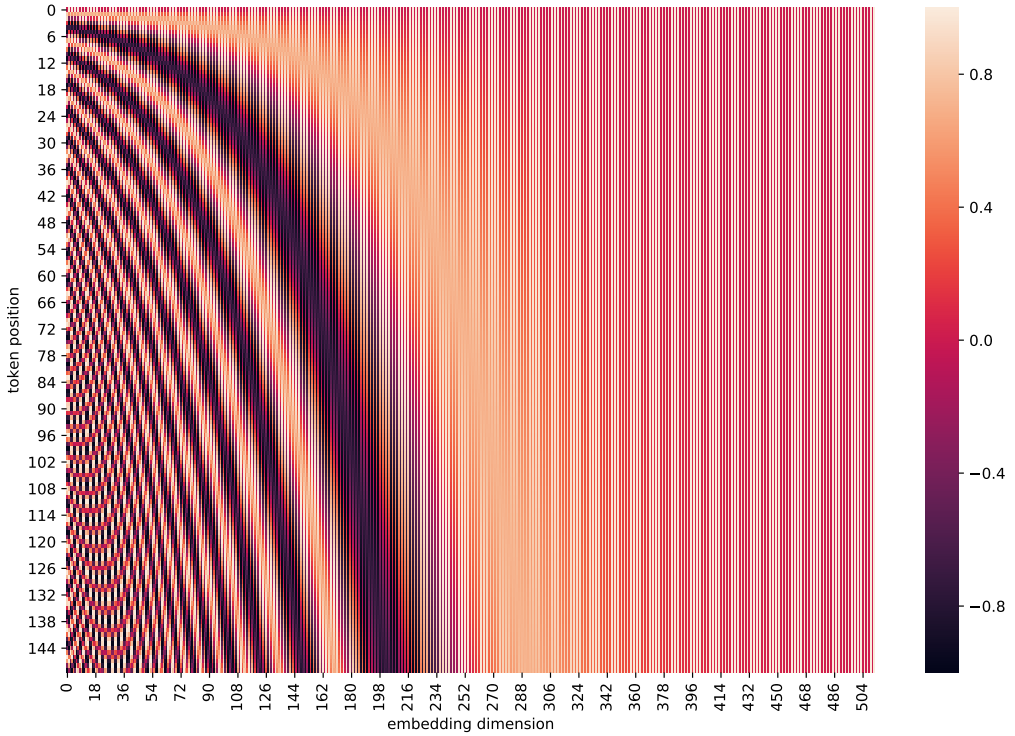


Figure 2.5: Positional embeddings with sine and cosine functions, embedding size $d = 512$.

To deal with this problem, the authors of the Transformer proposed added a positional embedding vector to each token embedding vector before feeding it to the network. This positional embedding vector is computed based on the token's position. Specifically, each element i of the vector for the token at position pos is computed by:

$$pos\_emb(pos, i) = \begin{cases} \sin\left(\frac{pos}{10000^{i/d}}\right), & \text{if } i \text{ is even} \\ \cos\left(\frac{pos}{10000^{(i-1)/d}}\right), & \text{if } i \text{ is odd} \end{cases} \tag{2.1}$$

in which d is the dimension of the embedding vector.

Figure 2.5 shows the values of the generated positional embeddings (dimension of 512) for token positions ranging from 1 to 150.

The authors also experimented with the learned positional embeddings, which were trained in the same manner as word embeddings. It was reported that the two methods yielded comparable results.

2.3.4 Model Architecture

All the features that we have discussed so far are used to construct the Transformer model (Figure 2.6).

The model takes the input token embeddings. After being combined with the corresponding positional encoding, they are passed through the encoder, which

is a stack of N multi-head attention layers with projection ("feed forward") layer on top of each. A similar process is used in the decoder. However, this time the multi-head attention is masked to avoid the attention layer looking into the future, i.e. it should only attend to the tokens translated up to that point. In addition, between this masked multi-head attention and the projection layer, there is another multi-head attention layer. This attention layer serves as the encoder-decoder attention in the `seq2seq` model. This is made possible by taking the queries from the output of the decoder's masked attention layer, while the keys and values come from the output of the encoder. The decoder's output is then passed through another linear projection layer and finally reaches a softmax layer to produce output probabilities.

It is also important to remind the reader that there are several residual connections in Figure 2.6 in which the information flow skips the multi-head attention or feed forward layer and is then recombined with the output of that layer in the "Add & Norm" layer. This "Add & Norm" layer also performs the layer-normalization step, which is shown to be useful for the `seq2seq` model [Ba et al., 2016].

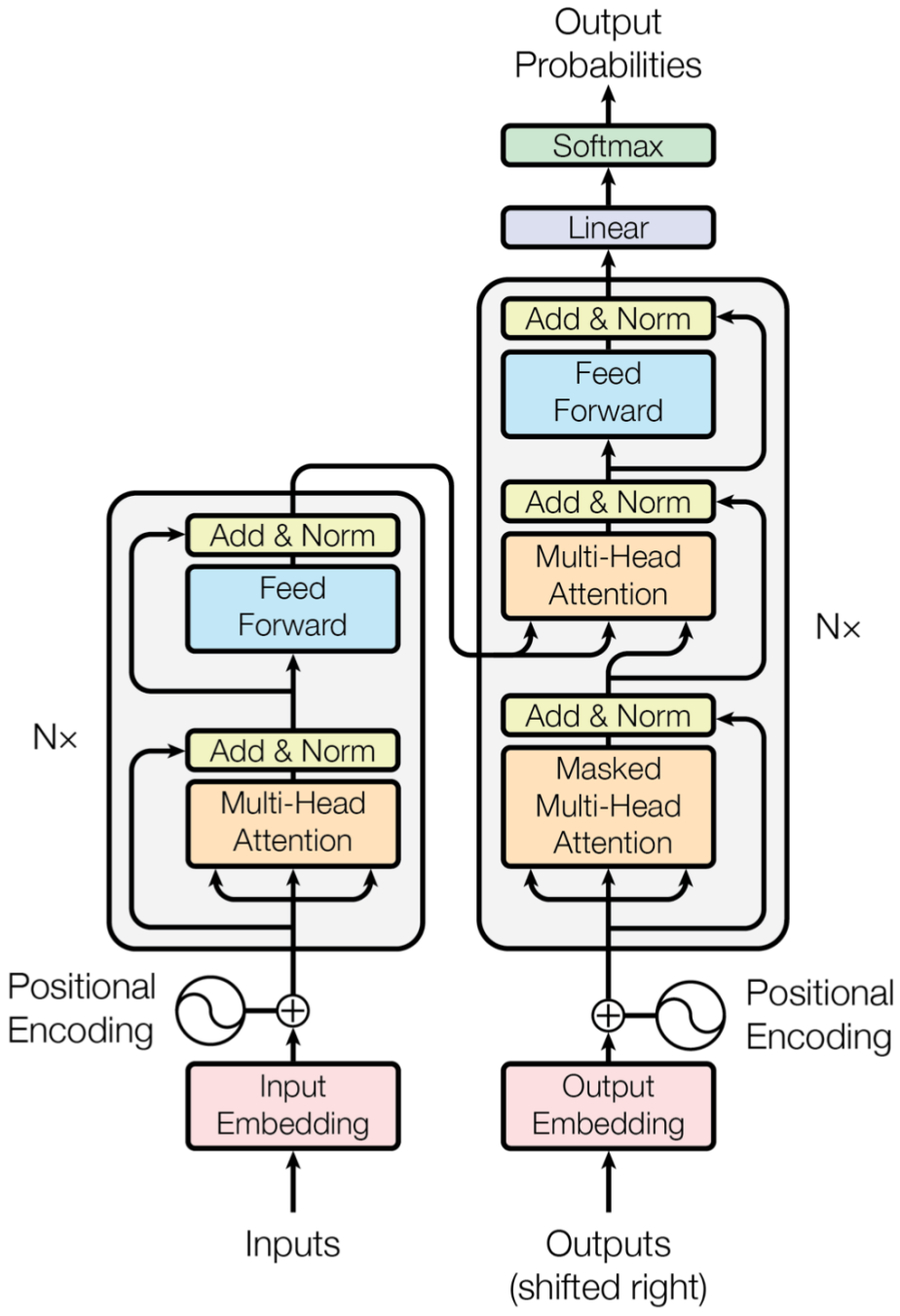


Figure 2.6: The Transformer model (adapted from Vaswani et al. [2017])

3. Enriching the Encoder by Targeting Self-Attention

In this chapter, we present our proposals to enrich the encoder. In contrast to Sennrich and Haddow, we do not simply feed the network with more information, e.g. POS tags or dependency labels, but directly instruct the self-attention mechanism to build upon these types of information. Section 3.1 discusses the first approach which introduces dependency-related attentional biases to the Transformer. In Section 3.2, we propose a novel specialized attention head guided by syntactic information.

3.1 Structured Attentional Bias

With this approach, we make the Transformer model aware of the source sentence structure by introducing bias terms which are similar to the relative position, but based on the dependency tree.

3.1.1 Relative Position

Let us recall the formula to compute the attention energy e_{ij} (before the softmax layer to get attention weight a_{ij}):

$$e_{ij} = \frac{1}{\sqrt{d_k}} x_i W^Q (x_j W^K)^\top \tag{3.1}$$

The energy e_{ij} is simply a dot product of the query $q_i = x_i W^Q$ (input x_i projected through W^Q) and the key $k_j = x_j W^K$ (input x_j projected through W^K) and divided by the square root of the attention hidden size d_k. The formula above assumes that the positional encoding has been included in the input. However, this positional encoding is generated from the absolute position of a token within the sentence. Shaw et al. [2018] proposed to impose the encoding of *relative position* in the attention layer instead by adding a bias term:

$$e_{ij} = \frac{1}{\sqrt{d_k}} x_i W^Q (x_j W^K + b^K_{ij})^\top \tag{3.2}$$

The bias term b^K_{ij} is the embedding vector of the relative position label between token i and j. Figure 3.1 illustrates this type of labels between token "is" and its neighbors in a sample sentence. These relative position labels are considered as discrete symbols, whose embeddings can be learned the same way as word embeddings.

Following Shaw et al., we also enrich the encoder's attention function by introducing structural biases. Instead of the relative position, we would like to use the information from the source-side dependency tree. We experiment with two forms of these labels: tree distance and tree traversal encoding.

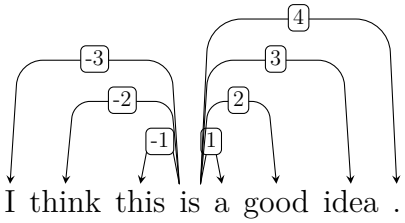


Figure 3.1: Relative position labels of the token *is* and its neighbors.

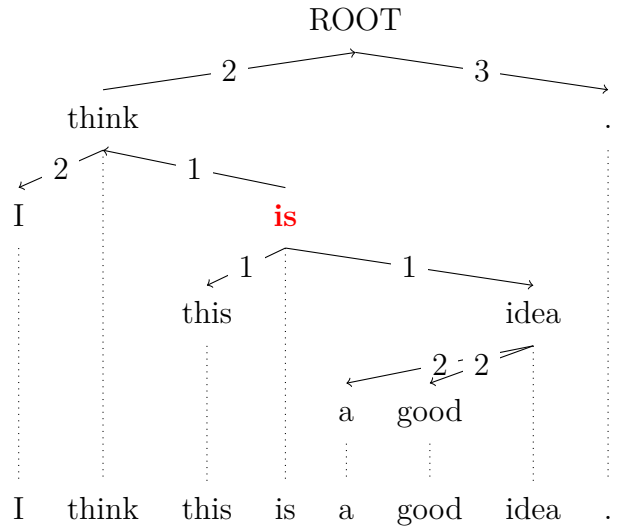


Figure 3.2: Example of tree distance labels of token *is* and the other tokens in a dependency tree. The arc label expresses the tree distance from *is* to the node at the tail of the arc.

3.1.2 Tree Distance

Our first proposal is the distance between two nodes in a dependency tree, i.e. the number of edges connecting the two nodes. Our model `TreeDistance` utilizes this distance embedding as the bias term t_{ij}^K instead of the relative position bias b_{ij}^K, or in a combination of both:

$$e_{ij} = \frac{1}{\sqrt{d_k}} x_i W^Q (x_j W^K + b_{ij}^K + t_{ij}^K)^\top \tag{3.3}$$

Figure 3.2 shows how we obtain the tree distance labels from a dependency tree. In this example, the distance between *is* and *think* is 1, because there is one edge connecting them. On the other hand, *is* and *a* is two edges apart, so the tree distance label is 2. It is obvious that this tree distance label is symmetrical, i.e. the tree distance between i and j is identical to the tree distance between j and i.

To be able to learn the embeddings, the model has to limit the maximum distance to maintain a fixed-size set of these labels. All token pairs that exceed this maximum distance are assigned a special label, which is similar to the out-of-vocabulary token (`<OOV>`).

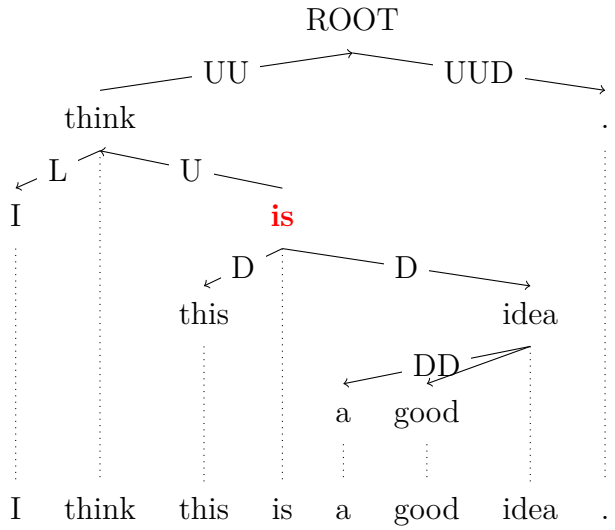


Figure 3.3: Example of tree traversal encodings of token *is* and the other tokens in a dependency tree. The arc label expresses the tree traversal encoding from *is* to the node at the tail of the arc.

3.1.3 Tree Traversal Encoding

We also expand the simple numerical tree distance above into the tree traversal encoding. This encoding does not only describe the distance but the path from one node to another. Our model `TreeTraversal` makes use of this encoding the same way the `TreeDistance` model utilizes the tree distance label.

Figure 3.3 elaborates our idea in a more intuitive way. Let us get back to the token *is*, to traverse from *is* to *ROOT*, we need to go up (U) twice. Hence, the traversal encoding from *is* to *ROOT* is "UU". It is important to note that while the tree distance is symmetrical, tree traversal encoding is not. This is because the direction of the path from *is* to *ROOT* is different from the path from *ROOT* to *is*.

In addition to the two characters 'U' (up) and 'D' (down), we also add two special characters to denote the path from the current node to its siblings: 'L' for its left siblings and 'R' for its right siblings. In Figure 3.3, the traversal encoding from *is* to I is "L" instead of "UD", because I is the left sibling of *is*. Sibling notation takes precedence over 'U' and 'D' but it is limited to siblings of the current node.

The maximum length of this traversal encoding ($max\_traversal$) is also restricted in order to have a fixed-size vocabulary of the traversal encodings. Moreover, the size of the vocabulary generated from our tree traversal does not grow exponentially to $max\_traversal$ because the encoding should be of the shortest path between two nodes in the tree. Hence, all possible encodings can be matched with the following regular expressions:

- **U*D*** : go all the way up (or not) and then down (or not).
- **LD*** : to the left sibling then down.
- **RD*** : to the right sibling then down.

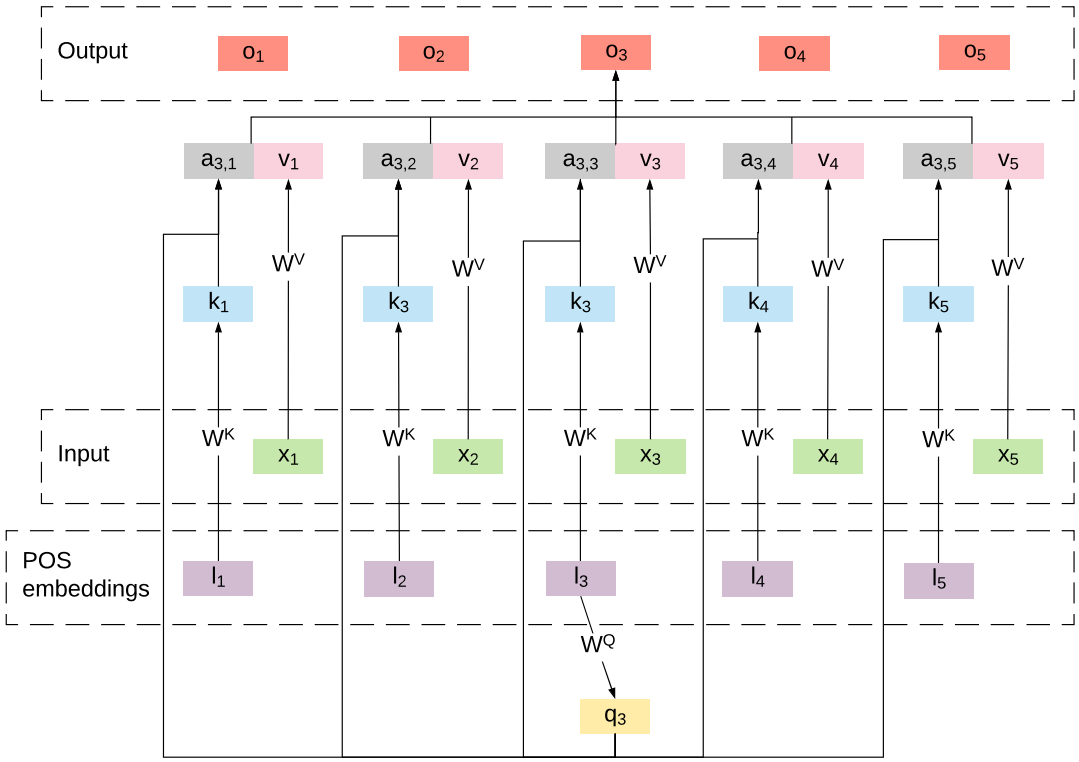


Figure 3.4: Specialized attention head with POS tag embeddings as keys and queries, computing the output o_3.

There is no reason to go down then up (e.g. "DDDDUU") because a shorter encoding with only 'D' characters can encode this path ("DD").

3.2 Specialized Attention Heads

In a normal self-attention layer (Section 2.3.1), the input x_i is projected through three matrices W^Q, W^K, W^V to obtain the corresponding key, query and value vectors. The reason for these projections, as discussed in the previous section, is to allow the key, query and value to come from various representation subspaces.

We hypothesize that it may be beneficial for the model that one of these vectors truly come from a known representation space even before the projection. To be specific, at the shallowest layer (layer 0) of the encoder, we propose the value to be projected from the token embedding (as in the standard Transformer), but the key and query should be projected from some linguistic label embeddings, e.g. POS tags or dependency labels.

As illustrated in Figure 3.4, the input x_i (token embedding on layer 0) is solely projected to the value vector v_i, while the query q_i and key k_i are obtained by projecting an additional input sequence l_i (POS embedding). Because the attention is guided by this supplementary information l_i, we would like to call it a "specialized attention head".

In our experiment, we only apply this specialized attention in one head of the shallowest layer in the encoder. In this chosen head, the model will calculate

the self-attention weights with keys k_i and queries q_i from one of the following linguistic labels:

- POS embeddings (`SpecPOS`, Figure 3.4).
- Dependency label embeddings (`SpecDep`, l_i is dependency label embedding instead).

4. Interpreting Self-Attention as Parse

In this chapter, we would like to suggest another set of approaches to multi-task training. Our proposed approaches attempt to jointly parse the source sentence and translate simultaneously by leveraging the self-attention mechanism in the encoder of the Transformer. Section 4.1 describes the model that is able to translate and parse the source sentence to produce source dependency tree at the same time. To examine the real benefit of dependency syntax, we let the model parse a simple sentence structure instead, see Section 4.2.

4.1 Self-Attention as Dependency Parse

Before presenting our proposal in Section 4.1.2, we would like to briefly introduce a neural dependency parsing model which is our main source of inspiration in Section 4.1.1.

4.1.1 Dependency Parser as Head Selection

The graph-based dependency parsers, which were discussed in Section 1.3.1, strive to produce a complete tree structure both during training and inference. There are also simpler solutions which learn to select the head token of the current token, i.e. head selection, independently. After the head selection process, a post-editing process handles the output to remove any existing cycle to form a proper tree. This approach has proven its effectiveness with the neural network model proposed by Dozat and Manning [2017].

The model, illustrated in Figure 4.1, utilizes two different LSTMs to learn the representation $h_i^{(arc-dep)}$ and $h_i^{(arc-head)}$ for each input x_i independently. This input x_i is a concatenation of the word embedding and POS embedding at position i in the input sentence. After that, an operation named *biaffine attention* is employed to produce S_{ij} which is the probability that token i is the head of token j:

$$S_{ij} = biaf(h_i^{(arc-head)}, h_j^{(arc-dep)}) \tag{4.1}$$

where function $biaf$ is a biaffine transformation. All the S_{ij} values form a matrix S which must be column-normalized. It is then compared against the gold parse tree, encoded as an adjacency matrix (Figure 4.2).

In the context of neural network, the concept of biaffine transformation can be defined, among the commonly used linear transformation and bilinear transformation, as follows:

Linear transformation is a function f of one variable x (n-dimensional), which can be written as

$$f(x) = x^\top W \tag{4.2}$$

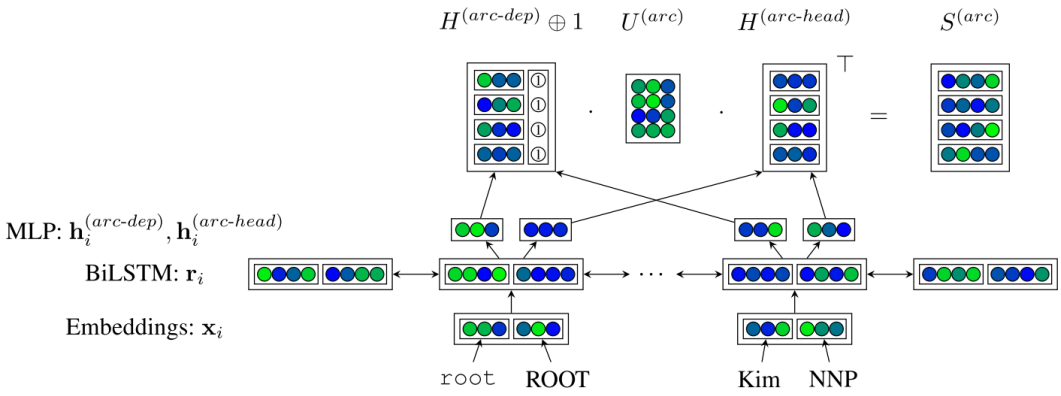


Figure 4.1: Neural dependency parser with deep biaffine attention (adapted from Dozat and Manning [2017])

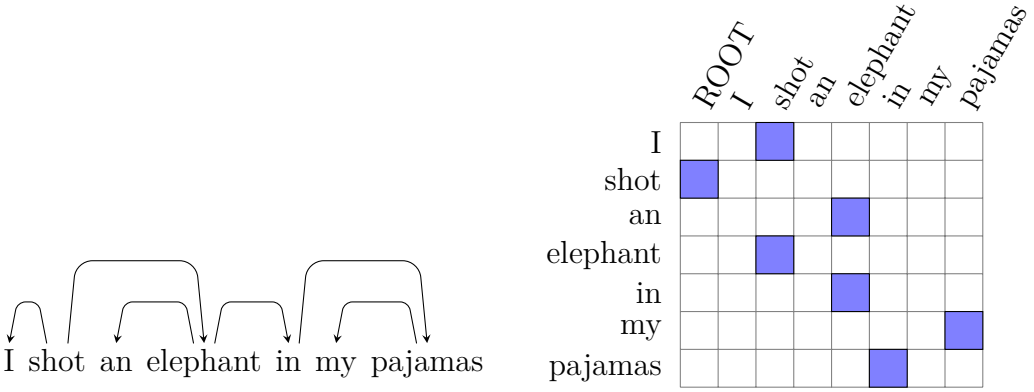


Figure 4.2: A dependency tree and its representation as an adjacency matrix (the columns represent the heads, the rows are dependents).

where W is the parameter of the transformation.

Bilinear transformation is a function f of two variables x_1 and x_2 (n_1-dimensional and n_2-dimensional, respectively), which can be written as

$$f(x_1, x_2) = x_1^\top W x_2 \tag{4.3}$$

where W is the parameter of the transformation. For any fixed x_1, $f(x_1, x_2)$ is linear in x_2 and for any fixed x_2, $f(x_1, x_2)$ is linear in x_1.

Biaffine transformation is a function f of two variables x_1 and x_2 (with dimension of n_1 and n_2, respectively), which can be written as

$$f(x_1, x_2) = x_1^\top W_1 x_2 + (x_1 \oplus x_2)^\top W_2 + b \tag{4.4}$$

where $x_1 \oplus x_2$ is the vector concatenation resulting in an $(n_1 + n_2)$-dimensional vector. W_1, W_2 and b are parameters with the dimensions of $n_1 \times n_2$, $n_1 + n_2$ and 1, in that order.

4.1.2 Parsing from Transformer's Self-Attention Weights

The construction of the S matrix above is very similar to the matrix of self-attention weights a_{ij} in the Transformer model. From this similarity, we speculate that the self-attentive architecture of Transformer NMT may have the capacity to learn dependency parsing and we only need to promote a little the particular linguistic dependencies captured in a treebank. Hence, we could attempt to simulate this parsing model with the self-attention layer in the Transformer model.

Figure 4.3 illustrates our joint model `DepParse`. The translation part is kept unchanged. The only difference is that we reuse one of the self-attention heads in the Transformer encoder and reinterpret it as if it was the dependency matrix S_{ij}. The training objective is combined and maximizes both the translation quality in terms of cross-entropy of the candidate translation and the unlabeled attachment score (UAS) of the proposed heads against the golden parse.[1]

The particular choice of the head which will serve as the dependency parser is arbitrary. Put differently, we constrain the Transformer model to use one of its heads to follow the given syntactic structure of the source sentence. It would be also possible to use the deep-syntactic parse of the sentence (the tectogrammatical layer as defined for the Prague Dependency Treebank, Hajič et al., 2006); we leave that for future work.

4.2 Self-Attention as Diagonal Parse

For contrast, we conduct an experiment with a simpler sentence structure, which we call the diagonal parse. In a diagonal parse, the dependency head of a token is simply the previous token (Figure 4.4).

Our model for the joint diagonal parsing and translation (`DiagonalParse`) is identical to the `DepParse` model, which has been described in Figure 4.3. We only need to use diagonal matrices during training, instead of the dependency matrices.

The main goal of this method is to examine whether or not the dependency structure really helps or any such constraining of the attention matrix can be beneficial, even a very simple one like the diagonal matrix.

[1] It should be noted that the dependency parses we use are actually automatic, produced by McDonald et al. [2005] parser incorporated in the Treex platform, formerly known as TectoMT [Popel and Žabokrtský, 2010]; `http://ufal.mff.cuni.cz/treex`.

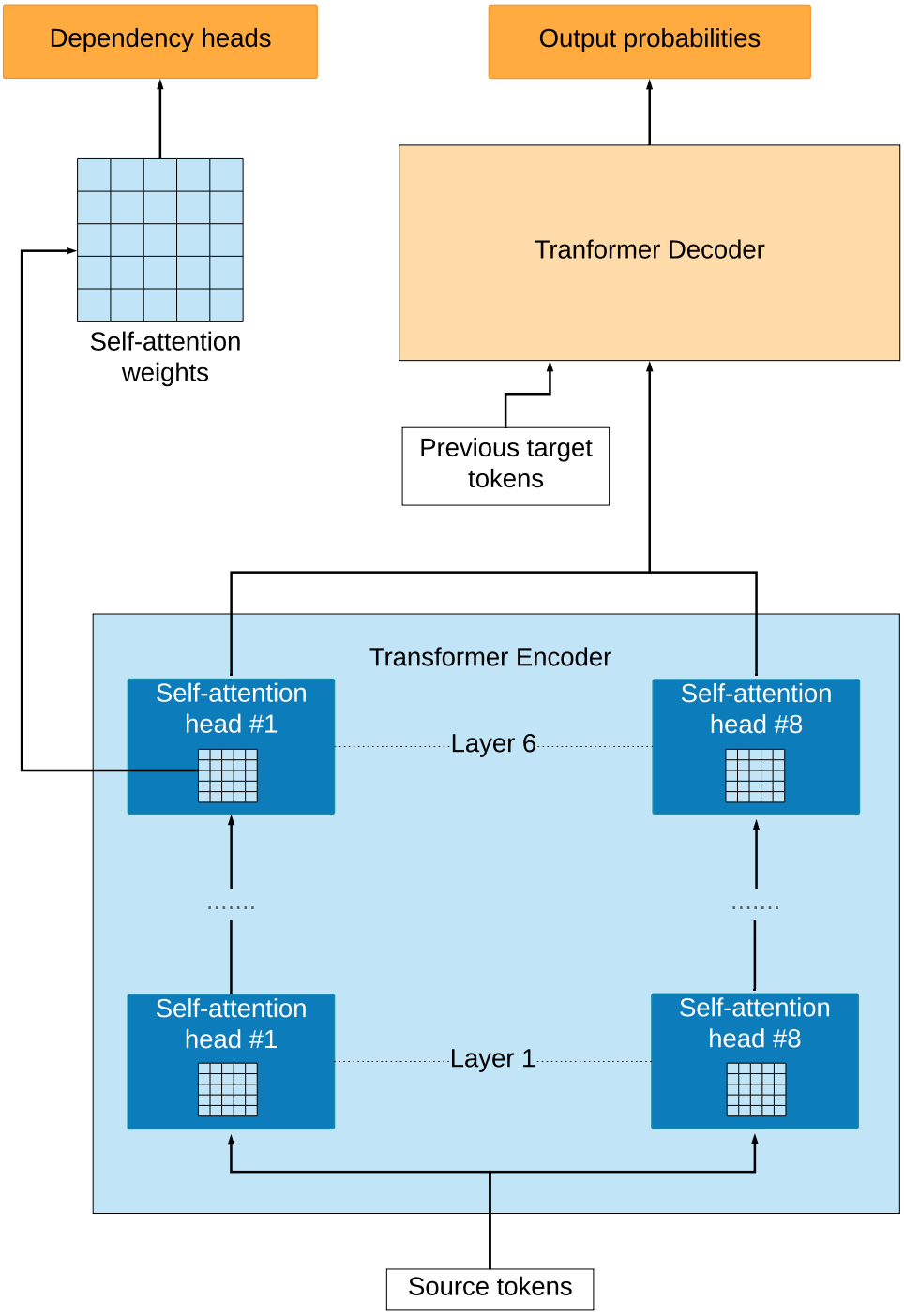


Figure 4.3: Joint dependency parsing and translation model (`DepParse` and `DiagonalParse`).

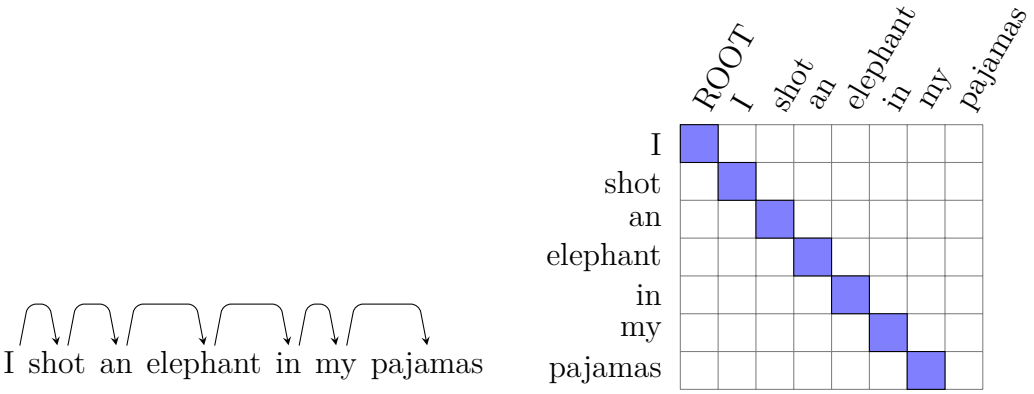


Figure 4.4: Dummy dependencies with diagonal matrix (the columns represent the heads, the rows are dependents).

5. Data and Experiment Setups

In this chapter, we present the datasets, environments and configurations used to conduct all of our experiments with the methods proposed in Chapters 3 and 4.

5.1 Data

We select Czech-to-English as our main translation task. Therefore, the primary dataset for our experiments is a subset of CzEng 1.7 [Bojar et al., 2016], which from now on will be mentioned as `cs2en`. To build `cs2en`, we select the CzEng's packages #00 to #08 for the training set, the first 1000 sentence pairs of package #09 to be the development set, and the next 10000 sentence pairs to be the test set. The reason for this subset is that we can have a standard average-size dataset and the training is more likely to converge within a reasonable amount of time. Basic statistic of this dataset is summarized in Table 5.2.

Among the available formats of CzEng, we choose the export format of this dataset for the smallest file size, ease of alignment between sentence pairs while still maintaining sufficient information. Each line in the export format file is an array of 15 blocks separated by the tab character. For more details about the data in these blocks, we encourage the reader to look them up in the dataset's home page.[1] For our experiments, we only need the 3^{th} block and the 7^{th} blocks, which are the a-layers (surface-syntactic trees) of the source sentence (Czech) and the target sentence (English). Each of these a-layers is in factored form (components delimited by the vertical bar '`|`') and consists of:

1. Word form.
2. Lemma.
3. Morphological tag (Czech) or POS tag (English).
4. Index in the sentence (absolute position).
5. Index of the governor (dependency head).
6. Syntactic function (dependency label).

Table 5.1 demonstrates an example of both a-layers from the source side and the target side, and also the information we can extract from them. Apart from the word forms of the sentences from both sides, we are only interested in the POS tag, the dependency head and the dependency label of each token in the source sentence.

For training and testing on parsing task, we use the same dataset whose source sentences were automatically annotated. The annotation provided in the CzEng release was originally created by Treex [Popel and Žabokrtský, 2010]. This annotation is based on the Prague Dependency Treebank (PDT, Hajič et al. [2006]), so we also use the PDT test set as a gold-annotated treebank to test our models.

[1] `http://ufal.mff.cuni.cz/czeng`

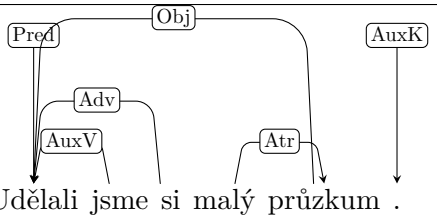
Source a-layer (factored form)	`Udělali|udělat_:W|VpMP---XR-AA---|1|0|Pred jsme|být|VB-P---1P-AA---|2|1|AuxV si|se_^(zvr._zájmeno/částice)|P7-X3----------|3|1|Adv malý|malý|AAIS4----1A----|4|5|Atr průzkum|průzkum|NNIS4-----A----|5|1|Obj .|.|Z:-------------|6|0|AuxK`
Source token	Udělali jsme si malý průzkum .
Source POS tag	VpMP---XR-AA--- VB-P---1P-AA--- P7-X3---------- AAIS4----1A---- NNIS4-----A---- Z:----------
Source dependency head	0 1 1 5 1 0
Source dependency label	Pred AuxV Adv Atr Obj AuxK
Source dependency tree	
Source diagonal head	0 1 2 3 4 5
Target a-layer (factored form)	`We|we|PRP|1|2|Sb did|do|VBD|2|0|Pred a|a|DT|3|5|AuxA little|little|JJ|4|5|Atr research|research|NN|5|2|Obj .|.|.|6|0|AuxK`
Target token	We did a little research .

Table 5.1: One sentence pair in the `cs2en` dataset.

For the multi-task models, we also experiment with a German-to-Czech corpus (`de2cs`) which was collected and processed from Europarl [Koehn, 2005] and OpenSubtitles2016 [Tiedemann, 2009] by Macháček [2018]. Similarly to CzEng, this dataset already includes auto-generated trees for source sentences. The parser used to generate such trees is the UDPipe framework [Straka and Straková, 2017] trained on Universal Dependencies 2.0 (UD, Nivre et al. [2017]). The reason for this dataset is that it is useful for us to have additional experiments with the new UD annotation, apart from the PDT. Nevertheless, the `de2cs` dataset comes with only CoNLL-U format[2] (tab separated table, Table 5.3) and needs to be converted to our export format.

The CoNLL-U format has ten columns:

1. ID: Word index, starting at 1.
2. FORM: Word form or punctuation symbol.
3. LEMMA: Lemma.

[2] `http://universaldependencies.org/format.html`

	de2cs	cs2en
Train set sentence pairs	8.8M	5.2M
Train set source tokens	89M	61M
Train set target tokens	78M	69M
Development set sent. pairs	news 2011: 3k	1k
Test set sentence pairs	news 2013: 3k	10k
Dependency parser generating the annotations	UD 2.0	Treex
Gold dependency treebank	de UD test	PDT test

Table 5.2: Data used in our experiments.

4. UPOS: Universal POS tag.
5. XPOS: Language-specific POS tag (morphological tags in case of Czech language).
6. FEATS: List of morphological features.
7. HEAD: Head of the current word (0 for root).
8. DEPREL: Universal dependency label.
9. DEPS: Enhanced dependency graph.
10. MISC: Other annotation.

ID	FORM	LEMMA	UPOS	XPOS	FEATS	HEAD	DEPREL	DEPS	MISC
1	“	“	PUNCT	$(	_	8	punct	_	SpaceAfter=No
2	Welcher	welche	PRON	PWS	Case=Nom|Gender=Masc|Number=Sing|PronType=Int	3	det	_	_
3	Kollege	Kollege	NOUN	NN	Case=Nom|Gender=Masc|Number=Sing	8	nsubj	_	_
4	hat	haben	AUX	VAFIN	Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin	8	aux	_	_
5	Ihnen	Sie|sie	PRON	PPER	Case=Dat|Person=2|Polite=Form|PronType=Prs	8	iobj	_	_
6	denn	denn	ADV	ADV	_	8	advmod	_	_
7	99	99	NUM	CARD	NumType=Card	8	obj	_	_
8	gesagt	sagen	VERB	VVPP	VerbForm=Part	0	root	_	SpaceAfter=No
9	?	?	PUNCT	$.	_	8	punct	_	SpaceAfter=No
10	”	”	PUNCT	$(	_	8	punct	_	_

Table 5.3: A source sentence in `de2cs`'s CoNLL-U format.

5.2 Experiment Setups

Experiments in this thesis were carried out with Tensor2Tensor[3] (T2T) version 1.5.6 at the *word level*, i.e. without using subword units [Sennrich et al., 2015]. We decided for this simplification for an easier alignment between the translation and parsing tasks. Because of operating on the word level, we limited the vocabulary size to be 50,000 tokens for each language.

The Transformer's hyperparameter set *transformer_base* [Popel and Bojar, 2018] was used as default for all model variants. From now on, we refer this to be one of the baseline model, named `Transformer base`. To be specific, some important hyperparameters in this setting are:

- Maximum sequence length = 256.
- 6 layers in the encoder.
- 6 layers in the decoder.
- 8 self-attention heads on each mutli-head attention layer.
- Embedding size = 512.
- Attention hidden size $d_k = 512/8 = 64$.
- Using layer normalization.
- Layer preprocess and postprocess dropout = 0.1.
- Attention dropout = 0.1.
- ReLU (rectified linear unit) dropout = 0.1.
- Learning rate warmup steps = 8000.
- Learning rate = 0.2.
- Adam optimizer:
  - $\epsilon = 1e^{-9}$.
  - $\beta_1 = 0.9$.
  - $\beta_2 = 0.997$.
- *Batch size = 3072.*

Batch size was the only hyperparameter that was set differently from the default *transformer_base*. The reason was we needed to fit all of our models to a single *GPU (graphical processing unit)* NVIDIA GTX 1080 Ti, which could not be done with the batch size of 4096. It is crucial for all experiments to have exactly the same batch size when training for comparable results.

Each variant of the Transformer model also has its own specific hyperparameters that we experimented with, namely:

[3] `https://github.com/tensorflow/tensor2tensor`

- **`Transformer relative`** - Transformer with relative position [Shaw et al., 2018], serves as our second baseline:
  - Remove positional encoding.
  - Maximum relative position = 20, i.e. if two tokens are more than 20 tokens apart, the relative position of this pair is a designated symbol similar to the out-of-vocabulary symbol.
- **`TreeDistance`** - Transformer with tree distance:
  - Maximum tree distance = 5 or 20 (similar to the maximum relative position).
  - Do or do not combine with the relative position.
  - Do or do not use the positional encoding.
- **`TreeTraversal`** - Transformer with tree traversal:
  - Maximum traversal path length = 10 (similar to the max relative position).
  - Do or do not combine with the relative position.
  - Do or do not use the positional encoding.
- **`SpecPOS`** & **`SpecDep`** - Transformer with specialized attention heads:
  - Type of information to guide the specialized attention head:
    - POS tags.
    - Dependency labels.
  - Do or do not combine with the relative position.
- **`DepParse`** & **`DiagonalParse`** - Leveraging self-attention weights of the Transformer's encoder to jointly translate and parse source sentences:
  - The layer from which the parse tree is demanded: 0 to 5.

For preprocessing, the only step needed to be done was to insert a dummy `<ROOT>` token at the beginning of every sentence, so that the selected self-attention matrix would be able to represent a dependency tree properly. This was only required for the `DepParse` model. After inference, this `<ROOT>` token was removed before the evaluation.

During inference, we used beam search with:

- Beam size = 4
- $\alpha = 0.6$
- Decoding batch size = 4

5.3 Evaluation

We used BLEU score to automatically evaluate translation task's performance. To be specific, we report cased BLEU using sacreBLEU,[4] a Python implementation of the official script in the Workshop of Machine Translation (WMT) shared task.[5] In addition, MT-ComparEval [Klejch et al., 2015] was also used to double check the score from sacreBLEU and to compute the statistical significance with bootstrap resampling [Koehn, 2004].

For the dependency parsing task, unlabeled attachment score (UAS) was employed to evaluate the output parse tree. UAS is in fact a precision score that measures the percentage of correctly predicted heads. The precision is also used to evaluate the diagonal parsing task.

To be able to observe the attention pattern of our models, we utilized the attention visualization notebook[6] in the T2T source code repository.

[4] https://github.com/mjpost/sacreBLEU

[5] ftp://jaguar.ncsl.nist.gov/mt/resources/mteval-v13a.pl

[6] https://github.com/tensorflow/tensor2tensor/blob/master/tensor2tensor/visualization/TransformerVisualization.ipynb

6. Results

In this chapter, we report the results from all of our experiments and additional observations from the neural network's behavior. It was reported that choosing a stopping criterion for NMT models is tricky [Popel and Bojar, 2018] depending on many aspects. We opted to stop training our models after 500,000 steps, at which the `Transformer base` was observed to show the sign of convergence.

6.1 Enriching Encoder

In this section, we report the results of our methods proposed in Chapter 3.

6.1.1 Tree Distance and Traversal

For the tree distance and tree traversal, it is shown that replacing the positional encoding or relative position with our proposed tree-related encodings does not help the translation. All of the proposed methods are worse than the baselines.

As listed in Table 6.1, on the test set, `Transformer base` and `Transformer relative` achieved 36.66 and 37.02 BLEU, respectively, whereas `TreeTraversal`, `TreeDistance` max 5 and max 20 only got 35.80, 33.13 and 35.50 BLEU, in that order. However, when the tree distance was combined with the positional encoding (`Transformer base`) or the relative position (`Transformer relative`), it yielded better results than both baselines (37.49 and 37.55 vs. 36.66 and 37.02). A similar gain was also seen in the case of tree traversal (37.80 vs. 37.02).

6.1.2 Specialized Attention Head

Also in Table 6.1, for our second approach in the direction of enriching the encoder, the `SpecPOS` model, whose one head in layer 0 was chosen to be dedicated as a specialized POS head, could not surpass the baselines (36.65 vs 36.66, 36.93 vs 37.02).

On the other hand, guiding this specialized attention head with dependency label embedding brought +1.06 BLEU over the `Transformer base` (37.72 vs 36.66). In addition, `SpecDep` when combining with `Transformer relative` also achieved +0.95 BLEU over the `Transformer relative` (37.97 vs 37.02).

6.2 Interpreting Self-Attention as Parse

This section presents the results of our joint models discussed in Chapter 4.

6.2.1 Diagonal Parsing

Before discussing the result of the parsing tasks (diagonal parsing and dependency parsing), we would like to note that a multi-task model usually requires much longer time to train compared to the single task model, e.g. twice the training time. However, all of our multi-task models discussed in this section and the following sections were also trained for only 500,000 steps in order to prove our

Model	Dev	Test
`Transformer base` (base)	37.28	36.66
`Transformer relative` (relative)	37.23	37.02†
`TreeDistance`, max 5	35.47	33.13
`TreeDistance`, max 20	34.45	35.50
`TreeTraversal`	35.21	35.80
(base) + `TreeDistance`, max 5	37.67	37.49$^{\ddagger\blacktriangledown}$
(relative) + `TreeDistance`, max 20	37.15	37.55$^{\ddagger\blacktriangledown}$
(relative) + `TreeTraversal`	38.22	**37.80**$^{\ddagger\blacktriangledown}$
(base) + `SpecPOS`	36.97	36.65
(relative) + `SpecPOS`	37.81	36.93†
(base) + `SpecDep`	37.53	37.72$^{\ddagger\blacktriangledown}$
(relative) + `SpecDep`	37.66	**37.97**$^{\ddagger\blacktriangledown}$

Table 6.1: Enriching encoder results on `cs2en`. Statistical significance marked as $\dagger$ ($p < 0.05$) and $\ddagger$ ($p < 0.01$) when compared to `Transformer base` and $\triangledown/\blacktriangledown$ when compared to `Transformer relative`.

	BLEU		Precision	
	Dev	Test	Dev	Test
`Transformer base`	37.28	36.66	–	–
Syntax demanded from head on layer 0	38.68	**38.14**	99.97	99.96
Syntax demanded from head on layer 1	39.11	38.06	99.99	**99.99**
Syntax demanded from head on layer 2	37.85	37.85	99.98	99.98
Syntax demanded from head on layer 3	37.93	37.70	99.97	99.98
Syntax demanded from head on layer 4	37.68	37.47	99.98	99.96
Syntax demanded from head on layer 5	37.53	37.54	99.96	99.95

Table 6.2: `DiagonalParse`'s results in translation (BLEU) and diagonal parsing (precision) on `cs2en`. All test BLEU improvements are statistically significant with $p < 0.01$ when compared to the `Transformer base`.

hypothesis that the Transformer already captures syntactic information. The comparison of training time will be reported in Section 6.3.

Table 6.2 reveals the result of our joint model which is capable of translating and parsing the source sentence to the diagonal matrix. The diagonal parsing precision is, as expected, very high, ranging from 99.95% to 99.99% on the test set. This joint model also outperformed the baseline in translation task with all its variants (BLEU scores vary from 37.47 to 38.14, compared to 36.66).

Moreover, these results form an observable pattern, in which the best result comes from the model which demands the linearized tree from the head on layer 0. Demanding that structure at deeper layers still helps but BLEU scores decrease. We believe a possible explanation for this pattern is because the diagonal matrix represents the relation between the preceding token and the current token. This is a very simple sentence structure and serves as an additional positional information to the absolute position embeddings. Therefore, the sooner the model is forced to recognize this positional information (via training the parsing task), the better it can learn to do translation.

6.2.2 Dependency Parsing

Similarly to Table 6.2, Table 6.3 presents the results of our joint dependency parsing and translation model.

Let us recall the discussion in Section 4.1.2 that the choice of the head from one layer which will serve as the dependency parse is arbitrary. However, selecting the layer matters. In the Transformer's encoder as used in our experiments, there are six multi-head attention layers. Table 6.3 shows the results of both tasks when different layers in the encoder were chosen to dedicate one of their self-attention heads to serve as the dependency matrix.

It is apparent from Table 6.3 that layer 0 (the first layer) was a too shallow layer to demand the syntax from. Demanding dependency syntax from this layer yielded undesirable results in both translation and parsing task. The BLEU score of 36.60 is no improvement over the baseline (36.66), while the UAS is at least 8% lower than other variants of the same model. We believe this result can be attributed to the assumption that the self-attention mechanism at this layer can only handle input word embeddings, which perhaps tend to concern more about lexical meaning than syntax. Therefore, it could not capture the complex dependency syntax, which is not as simple as the diagonal syntax mentioned in the previous section. On the other hand, layer 1 performed well on the parsing task (90.78%), and brought the best improvement to the translation task (38.01). The translation performance seems to be decreasing with deeper layers as well (38.01 - 37.87 - 37.67 - 37.60), as we already observed in the diagonal parsing. The reason, from our point of view, is also similar. The dependency syntax should be introduced to the model early enough in order to encourage better attention. At later stages, surface syntax becomes less relevant as the model builds its internal representation. Demanding syntactic interpretation from the deeper layers thus needlessly consumes the model capacity.

The performance on the dependency parsing task is shown to behave in the opposite direction. The UAS values in Table 6.3 are increasing from the shallower to the deeper layer (82.85 - 90.78 - 91.18 - 91.43 - 91.56). The highest UAS was achieved when the syntax is demanded from layer 4, while the model was still able to maintain a good translation performance. This pattern could suggest that when reaching to the deeper layers, the encoder has already learned a good representation for each token. By good, we mean that the relevant information from other tokens and the relation between those and the current token have been all brought together by the shallower layers. Therefore, at these deeper layers, the model can infer the dependency syntax easier and better.

It is important to remind the reader that the parsing results reported up to this moment were computed against the automatic parses of the parallel corpus. To have a better sense of how our models perform on the gold-annotated dataset, we tested our models with the PDT test set for Czech. The referential parser we used to compare against was from the winner of CoNLL Shared Task 2007 [Nivre et al., 2007], the latest available evaluation on the same dataset.

In addition to the PDT, we wanted to measure the performance on the new annotation convention — the Universal Dependencies 2.0. For this, we employed our additional dataset `de2cs`. As discussed in Section 5.1, this dataset includes the source trees generated by UDPipe. Therefore, we selected UDPipe to be the referential parser for this dataset.

	BLEU		UAS	
	Dev	**Test**	**Dev**	**Test**
`Transformer base`	37.28	36.66	–	–
Syntax demanded from head on layer 0	36.95	36.60	81.39	82.85
Syntax demanded from head on layer 1	38.51	**38.01**	90.17	90.78
Syntax demanded from head on layer 2	38.50	37.87	91.31	91.18
Syntax demanded from head on layer 3	38.37	37.67	91.43	91.43
Syntax demanded from head on layer 4	37.86	37.60	91.65	**91.56**
Syntax demanded from head on layer 5	37.63	37.67	91.44	91.46

Table 6.3: `DepParse`'s results in translation (BLEU) and dependency parsing (UAS) on automatically annotated data (`cs2en`). All test BLEU gains, except for layer 0, are statistically significant with $p < 0.01$ when compared to `Transformer base`.

Model	`de2cs`	`cs2en`
`Transformer base`	13.96	36.66
`DepParse`	$14.27^{\dagger}$	$38.01^{\ddagger}$

Table 6.4: BLEU scores on test sets for translation task. Statistical significance marked as † ($p < 0.05$) and ‡ ($p < 0.01$) when compared to `Transformer base`.

Table 6.4 presents the results, in which our proposed model outperformed the baseline in the translation task on both dataset (14.27 vs. 13.96 and 38.01 vs. 36.66).

When evaluated on the gold-annotated test sets, Table 6.5 shows that our model achieved a better result in German (de - 81.23 vs. 74.27), but worse in Czech (cs - 82.53 vs. 86.28). A possible excuse is that our model was trained using automatic treebanks, not the gold-annotated ones. We expect that the model could perform better after fine-tuning with the gold-annotated treebanks but leave this for future work.

6.2.3 Self-Attention Analysis

In this section, we further analyze self-attention layers of the `DepParse` and `DiagonalParse`'s encoders in the hope of a better understanding of the neural network we built.

Figure 6.1 presents the behavior of self-attention mechanism in each layer of our model that jointly translates and proposes dependency edges. In the figure, while each column represents one variant of our proposed model (except the first column which is the `Transformer base`), each row "Layer i" represents the $(i+1)$-th layer in that model. The histograms of self-attention weights were computed after concatenating attention weights from all heads on that layer, and for the first 100 sentences in the `cs2en` test set. In addition, the bin $[0.0, 0.1)$ has been removed for better visibility because most of the self-attention weights actually fell into this trivial bin.

As can be seen from the figure, the charts in the diagonal stand out, suggesting that the chosen layers have very sharp attention distributions, i.e. each head in

Model	de	cs
UDPipe 1.2 (de)	74.27	–
Nakagawa (2007)	–	86.28
`DepParse`	81.23	82.53

Table 6.5: UAS on the gold-annotated test sets for parsing task.

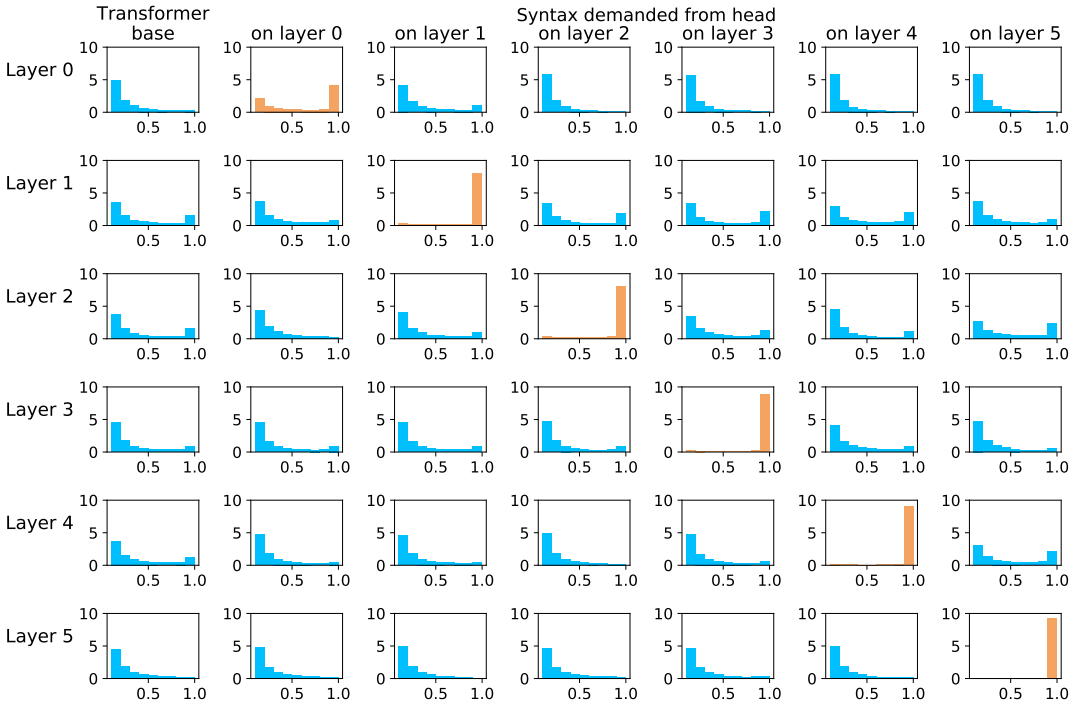
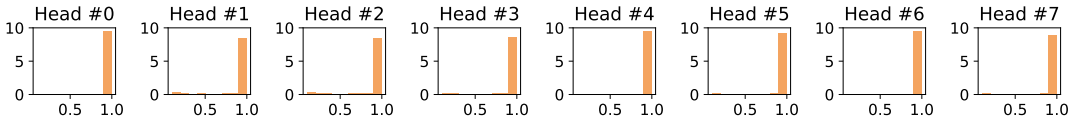


Figure 6.1: Histogram of normalized self-attention weights for each layers (all 8 heads) in the encoder.

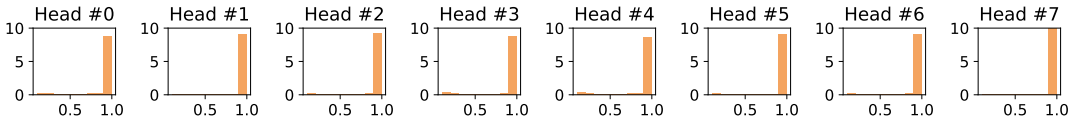
the layer attends to only one or two other positions in the previous layer. This behavior exists in all our multi-task models, except the "Syntax demanded from head on layer 0". As mentioned in Section 6.2.2, this particular model performed badly on both tasks. Our hypothesis is that this sharpness of attention due to our restricted self-attention helped the model to perform better.

One could argue that this sharp attention is expected for the head trained to produce syntactic tree. The interesting observation is that *all* heads in the respective layer also follow this pattern, as revealed in Figure 6.2.

In Figure 6.2, the histograms are computed separately for each head on layer 4 of the model where syntax was demanded from layer 4 (the bin $[0.0, 0.1)$ was also removed from the plot). For `DepParse` (Figure 6.2a), it is clear that not only the chosen head, but other heads in the same layer also have the sharp attention distribution. A similar pattern can also be observed in the case of `DiagonalParse` (Figure 6.2b). We believe the reason for this behavior lies in the concatenation and layer normalization after each multi-head attention layer. The layer normalization over all heads may introduce the sharpness from the chosen head to other heads in the same layer. We hypothesize that this caused a regularization in the network, which lead to our better performance in translation



(a) `DepParse` model.



(b) `DiagonalParse` model.

Figure 6.2: Histogram of self-attention weights for each head in layer 4 when demanding the parse tree from layer 4.

task. However, in the scope of this thesis, the identification of the exact reason is left for future work.

Figure 6.3 illustrates the sharpness studied above perhaps in a more intuitive way, by visualizing the attention weights for our sample sentence. The model generated this sample is `DepParse` that translates and produces dependency from one head on layer 4. It is obvious that the attention edges on layer 4 are sharper, each token concentrates on a smaller number of tokens in the sentence. On the other layers, each token attends to nearly all other tokens in the sentence, with a smaller weight for each attention edge.

6.3 Training Speed

Having documented that the Transformer NMT model has already captured dependency syntax, it is interesting to look at the training cost of the baseline and joint models. Our joint models derived from the Transformer were trained in a comparable time with the single-task `Transformer base`.

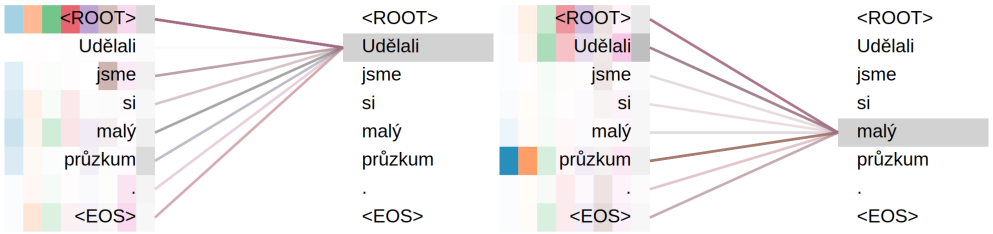
The training time (including internal evaluation every 1,000 steps) to reach 250,000 steps for the `Transformer base` was 1 day, 3 hours, 48 minutes on a single GPU NVIDIA GTX 1080 Ti. On average, our joint dependency parsing and translation model `DepParse` took only 13% longer than the `Transformer base` with a significant improvement in the translation task. Figure 6.4 also reveals the same for joint diagonal parsing and translation model `DiagonalParse`, which took 10% more time to train than the `Transformer base`.

The training times of encoder's enriched models with specialized attention heads are nearly identical to `Transformer base` and `Transformer relative`. The `SpecPOS` models derived from `Transformer base` and from `Transformer relative` took no extra time in comparison to those baselines. This also applies to the `SpecDep` models. However, they are different from the `SpecPOS` models in that they brought considerable improvements.

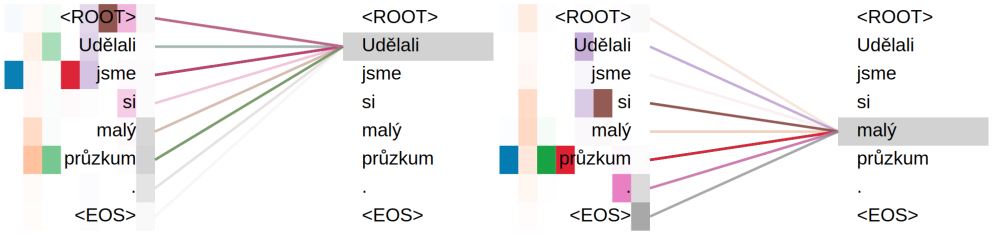
While having achieved similar results in translation task, the other set of methods enriching the encoder consumed more time to train. In detail, `TreeDistance` with maximum distance of 5 consumed 70% more of the time needed to train `Transformer base`, while `TreeTraversal` required 89% more. Combined with

positional encoding, `TreeDistance` max 5 took 96% more time in training, but this cost brought significant improvements over both `Transformer base` and `Transformer relative`. Finally, the total training times for the `TreeDistance` max 5 and the `TreeTraversal` when combined with relative positions are both 300% of the `Transformer base`'s training time (+200%). This huge amount of surplus training time was shown to yield improvements as well (Section 6.1).

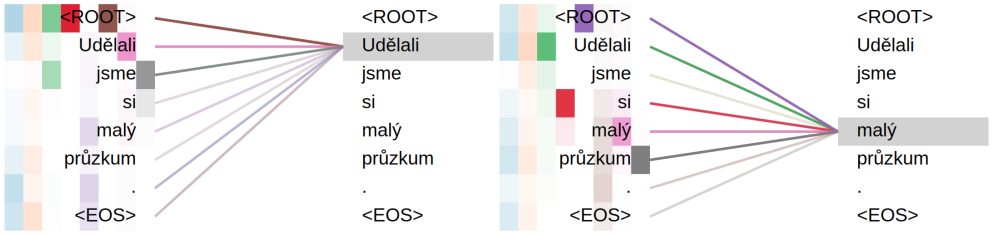
The extra amount of training time required by the family of `TreeDistance` and `TreeTraversal` models were caused by the tree distance and tree encoding matrix's generation, and also the multiplication of these matrices with key and query vectors. These computations are quadratic in the length of the input sentence and need to be done on every attention layer of the encoder. Interestingly, the joint models consumed only a little more time than `Transformer base` but brought improvements in translation tasks and a good parsing accuracy. This result was possible thanks to the fact that our joint models do not add any separate parser on top of the encoders as the common practice in multi-task models, but use the self-attention matrix as the near output layer, then top it with an $argmax$ layer for the head selection.



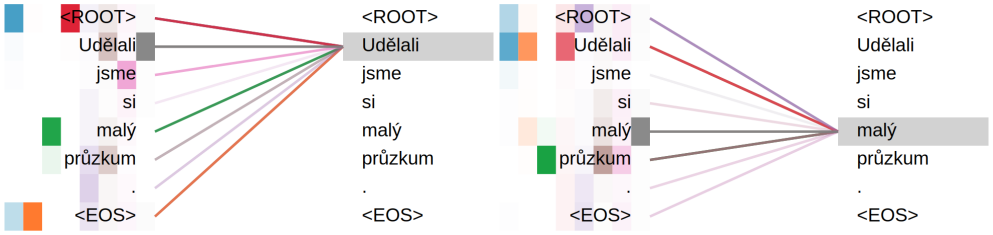
(a) Layer 0



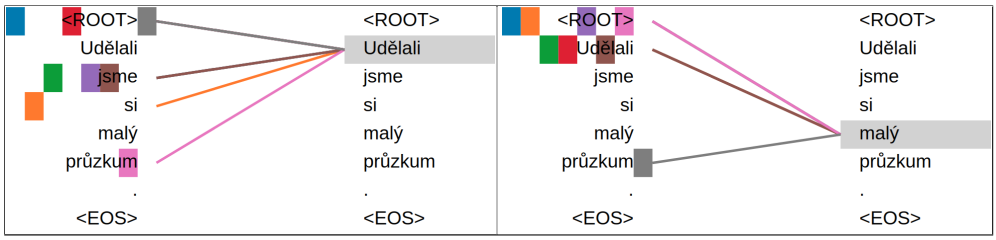
(b) Layer 1



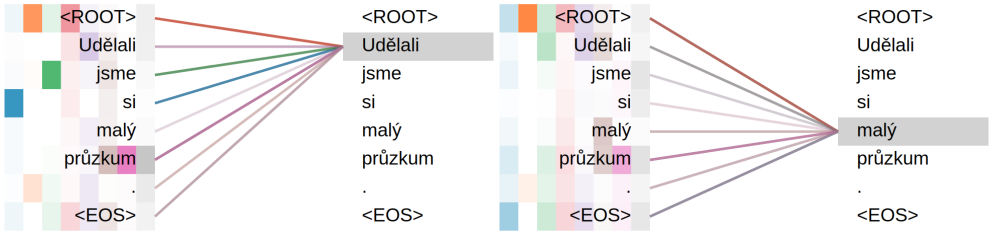
(c) Layer 2



(d) Layer 3



(e) Layer 4



(f) Layer 5

Figure 6.3: `DepParse` model with syntax demanded from the encoder's layer 4, displaying more focused attention at that layer (framed).

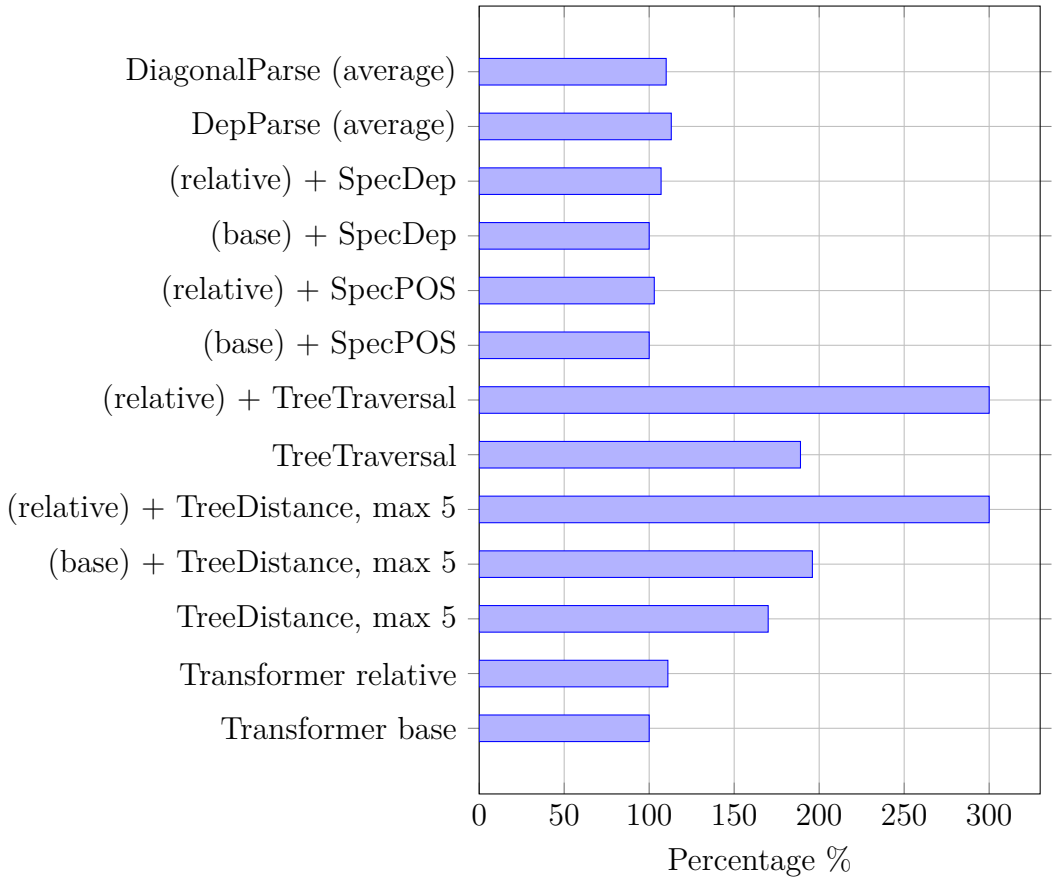


Figure 6.4: Training time to reach 250,000 steps on a single GPU NVIDIA GTX 1080 Ti (relative to `Transformer base`).

Conclusion

In this thesis, we proposed several methods to exploit the sentence structure in NMT by manipulation of the attention mechanism. While aiming at the same goal, the previous works focused mostly on the `seq2seq` model utilizing RNNs, we explore the state-of-the-art Transformer model which was built entirely on the attention mechanism. We speculate that this mechanism makes it easier for us to direct the neural network's behavior with additional syntactic information, e.g. the dependency tree.

We evaluated the models on Czech-to-English and German-to-Czech translations, both in relatively large data setting, against `Transformer base` (positional encoding) and `Transformer relative` (relative position).

The first set of methods attempted to enrich the encoder with source-side dependency tree. First, we replaced the positional encoding and relative position with our proposed tree distance and tree traversal encoding. BLEU scores of these models showed no improvement over the baselines. However, combining our proposals with the baselines reported +0.5 to +0.8 BLEU against both baselines. In addition, experiment results suggested that tree traversal works better than tree distance.

In this direction, we also proposed a specialized attention layer. The difference between this and the standard attention layer is that the key and query come from an additional input sequence containing linguistic information, either POS tags or dependency labels. The specialized POS head did not bring any improvement over baselines. On the other hand, specialized dependency head brought +0.95 to +1.06 compared against the baselines without this modification.

We further invented a novel component of the Transformer model for sentence structure parsing by promoting the interpretation of self-attention as dependency syntax, and showed that the Transformer model can be used as a precise parser. As suggested by the results, constraining self-attention to both true dependency as well as a simple diagonal matrix helped translation task at insignificant extra cost. The best model of `DepParse` and `DiagonalParse` achieved +1.35 and +1.48 BLEU improvements, respectively. The models also performed well on parsing tasks. `DepParse`'s best model achieved 91.56 UAS on the auto-generated treebanks, while `DiagonalParse`, unsurprisingly, obtained 99.99% accuracy. Furthermore, `DepParse` model performance is comparable to the referential parsers that were used to provide the parallel corpus with dependency trees.

To conclude our findings for the two main research questions of this thesis, it is clear from the results that enriching the Transformer with sentence structure can help. However, the Transformer model is in fact able to capture this type of linguistic information already on its own and the guidance through multi-task learning is needed only as a small push towards trees following the annotation rules.

Future Work

While this thesis has explored various possibilities of exploiting sentence structure in NMT, we believe that there is still a vast room for improvements, which

includes but is not limited to:

Fine-tuning the `DepParse` model with gold-annotated data. Leveraging the self-attention weights to do both parsing and translation outperformed the baseline translation model and was comparable to referential parsers. However, our model was trained on a synthetic treebank. Hence, we could also try to fine-tune our model with the gold annotated treebank, which we believe should lead to a better parsing performance.

Examining dependency on subword units. In this thesis, we were working on the word level, without subword units, for an easier alignment with the dependency structure of the sentence. Hence, all of the models had to face a serious out-of-vocabulary problem. We would like to examine various methods to push the dependency relation beyond the word level, to subword level. With that, we will be able to compare our proposed methods against the state-of-the-art translation models.

Bibliography

Roee Aharoni and Yoav Goldberg. Towards string-to-tree neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 2: Short Papers*, pages 132–140, 2017.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR*, 2015.

Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James R. Glass. What do neural machine translation models learn about morphology? In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 861–872. Association for Computational Linguistics, 2017.

Ondřej Bojar, Kamil Kos, and David Mareček. Tackling sparse data issue in machine translation evaluation. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 86–91. Association for Computational Linguistics, 2010.

Ondrej Bojar, Milos Ercegovcevic, Martin Popel, and Omar Zaidan. A grain of salt for the WMT manual evaluation. In Chris Callison-Burch, Philipp Koehn, Christof Monz, and Omar Zaidan, editors, *Proceedings of the Sixth Workshop on Statistical Machine Translation, WMT@EMNLP 2011, Edinburgh, Scotland, UK, July 30-31, 2011*, pages 1–11. Association for Computational Linguistics, 2011. URL `https://aclanthology.info/papers/W11-2101/w11-2101`.

Ondřej Bojar, Ondřej Dušek, Tom Kocmi, Jindřich Libovický, Michal Novák, Martin Popel, Roman Sudarikov, and Dušan Variš. CzEng 1.6: Enlarged Czech-English Parallel Corpus with Processing Tools Dockered. In Petr Sojka, Aleš Horák, Ivan Kopeček, and Karel Pala, editors, *Text, Speech, and Dialogue: 19th International Conference, TSD 2016*, number 9924 in Lecture Notes in Computer Science, pages 231–238, Cham / Heidelberg / New York / Dordrecht / London, 2016. Masaryk University, Springer International Publishing. ISBN 978-3-319-45509-9.

M. Asunción Castaño, Francisco Casacuberta, and Enrique Vidal. Machine translation using neural networks and finite-state models, 1997.

Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750, 2014.

Kehai Chen, Rui Wang, Masao Utiyama, Eiichiro Sumita, and Tiejun Zhao. Syntax-directed attention for neural machine translation. *CoRR*, abs/1711.04231, 2017.

Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561, Austin, Texas, November 2016. Association for Computational Linguistics. URL `https://aclweb.org/anthology/D16-1053`.

Trevor Cohn, Cong Duy Vu Hoang, Ekaterina Vymolova, Kaisheng Yao, Chris Dyer, and Gholamreza Haffari. Incorporating structural alignment biases into an attentional neural translation model. In *HLT-NAACL*, pages 876–885. The Association for Computational Linguistics, 2016.

Timothy Dozat and Christopher D. Manning. Deep biaffine attention for neural dependency parsing. In *ICLR 2017*, 2017.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In *HLT-NAACL*, pages 199–209. The Association for Computational Linguistics, 2016.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. Tree-to-sequence attentional neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho. Learning to parse and translate improves neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 2: Short Papers*, pages 72–78, 2017.

Hamidreza Ghader and Christof Monz. What does attention in neural machine translation pay attention to? In Greg Kondrak and Taro Watanabe, editors, *Proceedings of the Eighth International Joint Conference on Natural Language Processing, IJCNLP 2017, Taipei, Taiwan, November 27 - December 1, 2017 - Volume 1: Long Papers*, pages 30–39. Asian Federation of Natural Language Processing, 2017.

Yvette Graham, Timothy Baldwin, Alistair Moffat, and Justin Zobel. Continuous measurement scales in human evaluation of machine translation. In Stefanie Dipper, Maria Liakata, and Antonio Pareja-Lora, editors, *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse, LAW-ID@ACL 2013, August 8-9, 2013, Sofia, Bulgaria*, pages 33–41. The Association for Computer Linguistics, 2013. URL `http://aclweb.org/anthology/W/W13/W13-2305.pdf`.

Jan Hajič, Jarmila Panevová, Eva Hajičová, Petr Sgall, Petr Pajas, Jan Štěpánek, Jiří Havelka, Marie Mikulová, Zdeněk Žabokrtský, and Magda Ševčíková Razímová. Prague Dependency Treebank 2.0. LDC2006T01, ISBN: 1-58563-370-4, 2006.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory, 1995.

Ondřej Klejch, Eleftherios Avramidis, Aljoscha Burchardt, and Martin Popel. Mt-compareval: Graphical evaluation interface for machine translation development. *The Prague Bulletin of Mathematical Linguistics*, 104(1):63–74, 2015.

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

Philipp Koehn. Europarl: A Parallel Corpus for Statistical Machine Translation. In *Conference Proceedings: the tenth Machine Translation Summit*, pages 79–86, Phuket, Thailand, 2005. AAMT, AAMT.

Kamil Kos and Ondřej Bojar. Evaluation of machine translation metrics for czech as the target language. In *Prague Bulletin of Mathematical Linguistics*, pages 92–135, 2009.

An Nguyen Le, Ander Martinez, Akifumi Yoshimoto, and Yuji Matsumoto. Improving sequence to sequence neural machine translation by utilizing syntactic dependency information. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing, IJCNLP 2017, Taipei, Taiwan, November 27 - December 1, 2017 - Volume 1: Long Papers*, pages 21–29, 2017.

Junhui Li, Deyi Xiong, Zhaopeng Tu, Muhua Zhu, Min Zhang, and Guodong Zhou. Modeling source syntax for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 688–697, 2017.

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *EMNLP*, pages 1412–1421. The Association for Computational Linguistics, 2015.

Dominik Macháček. Enriching neural mt through multi-task training. Master's thesis, Charles University in Prague, 6 2018.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *COMPUTATIONAL LINGUISTICS*, 19(2):313–330, 1993.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-Projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of HLT/EMNLP 2005*, October 2005.

Igor' Aleksandrovič Mel'čuk. *Dependency syntax: theory and practice*. SUNY press, 1988.

Ramon P Neco and Mikel L Forcada. Asynchronous translations with recurrent neural nets. In *Neural Networks, 1997., International Conference on*, volume 4, pages 2535–2540. IEEE, 1997.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan T. McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. The conll 2007 shared task on dependency

parsing. In *EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic*, pages 915–932, 2007. URL http://www.aclweb.org/anthology/D07-1096.

Joakim Nivre, Željko Agić, Lars Ahrenberg, Lene Antonsen, Maria Jesus Aranzabe, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Eckhard Bick, Cristina Bosco, Gosse Bouma, Sam Bowman, Aljoscha Burchardt, Marie Candito, Gauthier Caron, Gülşen Cebiroğlu Eryiğit, Giuseppe G. A. Celano, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Silvie Cinková, Çağrı Çöltekin, Miriam Connor, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Marhaba Eli, Ali Elkahky, Tomaž Erjavec, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčová, Petter Hohle, Radu Ion, Elena Irimia, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Hiroshi Kanayama, Jenna Kanerva, Tolga Kayadelen, Václava Kettnerová, Jesse Kirchner, Natalia Kotsyba, Simon Krek, Sookyoung Kwak, Veronika Laippala, Lorenzo Lambertino, Tatiana Lando, Phuong Lê H`ông, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Anna Missilä, Verginica Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Shunsuke Mori, Bohdan Moskalevskyi, Kadri Muischnek, Nina Mustafina, Kaili Müürisep, Pinkey Nainwani, Anna Nedoluzhko, Luong Nguy˜ên Thị, Huy`ên Nguy˜ên Thị Minh, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Petya Osenova, Lilja Ovrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Martin Popel, Lauma Pretkalniņa, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Alexandre Rademaker, Livy Real, Siva Reddy, Georg Rehm, Larissa Rinaldi, Laura Rituma, Rudolf Rosa, Davide Rovati, Shadi Saleh, Manuela Sanguinetti, Baiba Saulīte, Yanin Sawanakunanon, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Lena Shakurova, Mo Shen, Atsuko Shimada, Muh Shohibussirri, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Antonio Stella, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Takaaki Tanaka, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Gertjan van Noord, Viktor Varga, Veronika Vincze, Jonathan North Washington, Zhuoran Yu, Zdeněk Žabokrtský, Daniel Zeman, and Hanzhi Zhu. Universal dependencies 2.0, 2017. URL http://hdl.handle.net/11234/1-1983. LIN-

DAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL `https://doi.org/10.3115/1073083.1073135`.

Martin Popel and Ondřej Bojar. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):43 – 70, 2018.

Martin Popel and Zdeněk Žabokrtský. TectoMT: Modular NLP framework. In Hrafn Loftsson, Eirikur Rögnvaldsson, and Sigrun Helgadottir, editors, *Lecture Notes in Artificial Intelligence, Proceedings of the 7th International Conference on Advances in Natural Language Processing (IceTAL 2010)*, volume 6233 of *Lecture Notes in Computer Science*, pages 293–304, Berlin / Heidelberg, 2010. Iceland Centre for Language Technology (ICLT), Springer. ISBN 978-3-642-14769-2.

Maja Popovic. chrf: character n-gram f-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation, WMT@EMNLP 2015, 17-18 September 2015, Lisbon, Portugal*, pages 392–395, 2015. URL `http://aclweb.org/anthology/W/W15/W15-3049.pdf`.

Rico Sennrich and Barry Haddow. Linguistic input features improve neural machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 1, Research Papers*, volume 1, pages 83–91, 2016.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *HLT-NAACL*. The Association for Computational Linguistics, 2018.

Xing Shi, Inkit Padhi, and Kevin Knight. Does string-based neural MT learn source syntax? In Jian Su, Xavier Carreras, and Kevin Duh, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1526–1534. The Association for Computational Linguistics, 2016.

Miloš Stanojević and Khalil Sima'an. Fitting sentence level translation evaluation with many dense features. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 202–206, Doha, Qatar, October 2014. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D14-1025`.

Milan Straka and Jana Straková. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada, August 2017. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/K/K17/K17-3009.pdf`.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *ACL (1)*, pages 1556–1566. The Association for Computer Linguistics, 2015.

Lucien Tesnière. Eléments de syntaxe structurale. *Klincksieck, Paris*, 1959.

Jörg Tiedemann. News from OPUS - A collection of multilingual parallel corpora with tools and interfaces. In *Proc. of RANLP*, volume V, pages 237–248. John Benjamins, Amsterdam/Philadelphia, Borovets, Bulgaria, 2009. ISBN 978 90 272 4825 1.

Ke Tran and Yonatan Bisk. Inducing grammars with and for neural machine translation. *arXiv preprint arXiv:1805.10850*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6000–6010, 2017.

List of Figures

List of Tables

List of Abbreviations

BLEU	bilingual evaluation understudy
cs	Czech
cs2en	Czech-to-English
de	German
de2cs	German-to-Czech
en	English
GPU	graphical processing unit
MT	machine translation
NMT	neural machine translation
PDT	Prague Dependency Treebank
POS	part of speech
RBMT	rule-based machine translation
ReLU	rectified linear unit
RNN	recurrent neural network
seq2seq	sequence-to-sequence
SMT	statistical machine translation
T2T	Tensor2Tensor
UD	Universal Dependencies
WMT	Workshop on Machine Translation