Errata

I present corrections for the submitted version. All changes are enumerated here. I also included the whole text (corrected version) as it could be possibly more comfortable for readers.

Table 2.6

Table 2.6 was supplemented by macro-F1 scores, if available. There were also some mistakes in reported related work, namely some reported numbers were macro-F1, not weighted. I also added other results from (Klouda et al., 2019), because the reported ones were worse. Finally, I added (Habernal et al., 2013), as it is the original paper for used datasets.

Table 2.9

Table 2.9 is changed. There is new column with macro-F1 for all experiments (supplemented and corrected). I also removed experiment 42 as it was just an additional experiment does not belonging among others.

Overflowing Picture 1.8

Accidental overflowing of picture 1.8 out form the page is corrected in this version and I also include the picture here:

Missing Picture in Section 2

I revealed a reference for a missing picture of directory structure on page 61 and I think it is no longer needed as I simplified the structure in both GitHub and in attachments to the work.

Attachments Description

Originally, I included also the references to models in GitHub Large Files Storage. Unfortunately I realized later that I exceeded the quota, so they will not work. References to models will be available on the main page of GitHub repository. I also included unfinished version of working example `play_with_models.ipynb`. Working version will be also available on GitHub.

dataset	models	Acc	F1-w	F1-m
All Czech	baseline	82.00	70.00	-
	(Kyselý, 2017)	*67.82*	*67.00*	-
	best(16)	**84.04**	**83.86**	**80.84**
csfd	baseline	69.07	69.00	-
	Czert	-	-	*84.79*
	(Habernal et al., 2013)	-	-	79.00
	(Kyselý, 2017)*	*71.34*	*71.00*	-
	best(16)	84.02	84.00	-
	best(69)	**84.89**	**84.87**	**84.83**
mall	baseline	84.72	83.00	-
	(Kyselý, 2017)	*82.52*	*81.00*	-
	(Habernal et al., 2013)	-	-	75.00
	(Klouda et al., 2019)(Bert)	81.00	79.00	-
	(Klouda et al., 2019)(SVM)	84.00	82.00	-
	best(16)	84.40	84.00	-
	best(63)	**84.60**	**84.14**	**76.85**
facebook	baseline	67.30	63.00	-
	RobeCzech	-	-	*80.13*
	(Habernal et al., 2013)	-	-	69.00
	XLM-RoBERTa**	-	-	***82.29***
	Czert	-	-	*76.55*
	(Kyselý, 2017)	*71.62*	*71.00*	-
	best(16)	75.00	74.98	-
	best(45)	**81.80**	**81.65**	80.11

Table 1: Best results for all datasets and a comparison to previous work. Best(16) is a best model for joint dataset and best(x) is always the best model for respective dataset. Numbers in italics are from related work. Related work results except *Czert* are 10-fold crossvalidation results. $\star$ (Kyselý, 2017) performs only sentence-level classification. $\star\star$ This is the large XLM-RoBERTa model from Straka et al. (2021), which is four times larger than the BERT base model.

Formal Mistakes

In the version presented in this errata, I also corrected some typographical mistakes, e.g., missing citation on page 40 (due to an error in latex source, too wide text on page 43 or too high table 2.9. These were only small formal corrections, which only produce more aesthetic text, and does not change substantially the work.

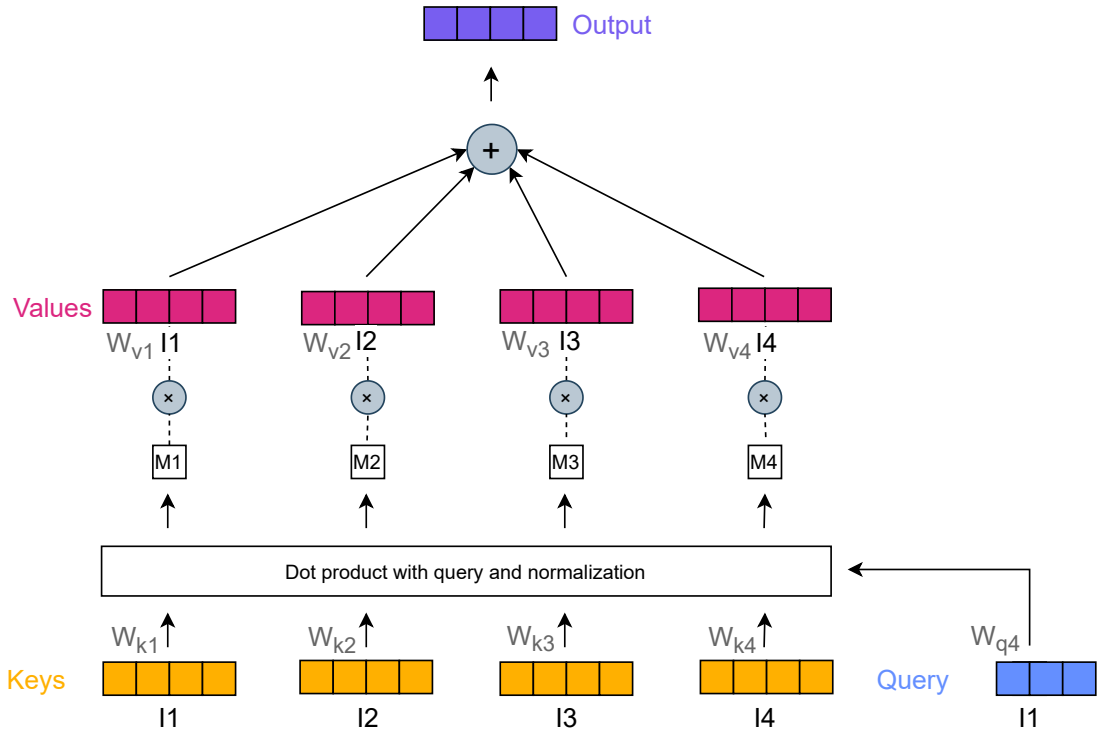
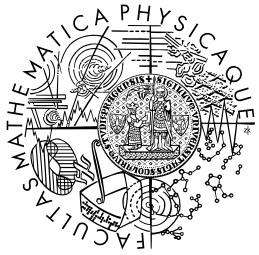


Figure 1: Self-attention mechanism scheme for one selected query vector. The result is an embedding, which is improved by the context of the word. This picture illustrates the result for the embedding of the first word (l1) in a four-word long text. Input words are denoted as l1 to l4. Keys, values, and a query are all computed by multiplying the input embeddings by their respective weights (W_k, W_v, and W_q) before any other operation with them. These weights are trained during learning. Dot products between every word and every query are computed. The result is a number for every input word, so four numbers at the end. These numbers are normalized, so the sum of them is equal to 1. These numbers serve as a weight (M_x), which indicates the relationship between the query and every other word. The resulting better embedding for the query is then obtained as a sum of the word embeddings weighted by these obtained weights.

MODEL		EXPE	LAYERS	LRTYPE	Acc	F1-w	F1-m
1	mBERT	czech	four	isrd	80.89	80.62	76.89
2		zero			49.51	44.67	35.91
3		eng			81.17	80.90	77.45
4		czech		cos	82.56	82.35	79.10
5		zero			53.41	47.64	38.49
6		eng			82.55	82.37	79.12
13	RoBECzech	czech	four	isrd	81.17	80.90	79.65
14		zero			55.31	48.26	38.79
16		czech		cos	84.04	83.86	80.72
17		zero			57.64	48.79	39.03
19	mBERT	czech	att	isrd	81.61	81.43	78.02
20		zero			53.92	47.55	38.23
21		eng			81.79	81.32	77.78
22		czech		cos	82.62	82.42	79.11
23		zero			51.99	46.63	37.59
24		eng			82.59	82.36	79.08
31	RoBECzech	czech	att	isrd	83.26	83.18	80.06
32		zero			58.36	50.40	40.89
34		czech		cos	83.88	83.68	80.57
35		zero			58.13	50.89	35.71
37	mBERT	facebook	four	isrd	75.30	74.97	35.71
38				cos	76.20	75.89	73.32
41	RoBECzech			isrd	80.10	79.87	77.98
43				cos	81.50	81.37	79.90
44					81.00	80.78	79.02
45					81.80	81.65	80.11
46	mBERT		att	isrd	76.40	75.67	72.68
47				cos	77.20	76.83	74.20
50	RoBECzech			isrd	79.60	79.07	76.78
51				cos	80.60	80.38	78.76
52	mBERT	mall	four	isrd	82.80	82.80	74.94
53				cos	84.27	83.88	76.48
56	RoBECzech			isrd	83.17	83.37	76.00
57				cos	84.73	84.30	76.95
58	mBERT		att	isrd	83.02	82.90	75.36
59				cos	84.04	83.61	75.94
62	RoBECzech			isrd	84.08	83.88	76.18
63				cos	84.60	84.14	76.85
64	mBERT	csfd	four	isrd	80.77	80.83	80.79
65				cos	82.04	82.04	82.01
68	RoBECzech			isrd	83.06	83.05	83.00
69				cos	84.89	84.87	84.83
70	mBERT		att	isrd	81.63	81.60	81.57
71				cos	82.20	82.19	82.16
74	RoBECzech			isrd	83.13	83.18	83.13
75				cos	84.32	84.32	84.28

Table 2: This table presents complete results on the sentiment task. Presented metrics are accuracy, macro-F1, and weighted-F1.



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Petra Vysušilová

Czech NLP with Contextualized Embeddings

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	RNDr. Milan Straka, Ph.D.
Study programme:	Computer science
Study branch:	Artificial intelligence

Prague 2021

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.

In ..... date ..... signature of the author

Firstly, I would like to express my gratitude to my supervisor, Milan Straka, for answering even the dumbest questions with patience and kindness, helping me to debug tricky errors as well as the obvious ones, and always being cheerful and full of enthusiasm. I learned so much during the work on this thesis.
This thesis would not exists without my loving husband and his continuous support. I would also like to thank my s[o/u]n, who didn't let me program all day without interruption and go crazy.
I could never start studying without my mom and grandma, who raised me and gave me the best.
Last but not least, Thank God, who created such a beautiful world.

P.S.: My husband wants me to write here that he is amazing (and he truly is).

Title: Czech NLP with Contextualized Embeddings

Author: Bc. Petra Vysušilová

Institute: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Milan Straka, Ph.D., Institute of Formal and Applied Linguistics

Abstract: With the increasing amount of digital data in the form of unstructured text, the importance of natural language processing (NLP) increases. The most successful technologies of recent years are deep neural networks. This work applies the state-of-the-art methods, namely transfer learning of Bidirectional Encoders Representations from Transformers (BERT), on three Czech NLP tasks: part-of-speech tagging, lemmatization and sentiment analysis. We applied BERT model with a simple classification head on three Czech sentiment datasets: mall, facebook, and csfd, and we achieved state-of-the-art results. We also explored several possible architectures for tagging and lemmatization and obtained new state-of-the-art results in both tagging and lemmatization with fine-tunning approach on data from Prague Dependency Treebank. Specifically, we achieved accuracy 98.57% for tagging, 99.00% for lemmatization, and 98.19% for joint accuracy of both tasks. Best models for all tasks are publicly available.

Keywords: Natural Language Processing BERT Bidirectional Encoder Representations from Transformers Part-of-Speech Tagging Lemmatization Sentiment Analysis

Contents

List of Abbreviations

AI Artificial Intelligence. 5

BERT Bidirectional Encoder Representations from Transformers. 27, 64

GRU Gated Recurrent Unit. 47

LSTM Long Short-Term Memory. 47

NLP Natural Language Processing. 1, 4–7, 10, 11, 13–15, 17, 24–27, 33, 35

POS part-of-speech. 4–7, 42, 43, 64

RNN Recurrent Neural Network. 45, 47

SOTA state-of-the-art. 4, 43, 44, 48, 52, 64

Introduction

Motivation

People think and communicate in natural languages. They express their opinions, share information and feelings, or persuade others about their ideas, all in natural languages. In the current era of digital technologies, all this information (sadly even information people do not share consciously and with the awareness of the potential risks) are available online. The amount of data is so enormous that it is not in human power to sort and use them, and that is when Natural Language Processing (NLP) is a necessary step for further processing by computers. For these reasons, many NLP use cases exist, for example extracting opinions about new products (e.g., via sentiment analysis or topic modeling), using chatbots instead of paying employees in a call center, voice assistance for people with hearing or vision impairment, filtering spam from email, summarizing the content of papers or finding answers in texts. In recent years, neural networks have achieved great success in many areas, for example computer vision, speech recognition, or marketing, and they get to all areas of research and industry. This work applies the most successful deep learning NLP methods of recent years to Czech Natural Language Processing tasks: part-of-speech (POS) tagging, lemmatization and sentiment analysis. First two tasks are low-level tasks used as a part of data processing pipeline for almost every other NLP tasks. In contrast, sentiment analysis is an example of a task interesting for end user outside the computer science field and this task also demonstrates the help of used models in getting rid of complicated hand-crafted architectures. Tasks were chosen from both semantics and syntax to show how pre-trained multilingual language models can help with different types of Natural Language Processing (NLP) tasks

Goals of this work

This work aims to improve selected Natural Language Processing (NLP) tasks for Czech with the use of recently published state-of-the-art (SOTA) techniques, namely transfer learning of (possibly multilingual) bidirectional language models. This work uses two pre-trained multilingual models (BERT (Devlin et al., 2019) and XLM-RoBERTa (Conneau et al., 2019)), that were trained in many languages including Czech, and a monolingual Czech variant of RoBERTa called RobeCzech (Straka et al., 2021). Selected tasks are tagging, lemmatization, and sentiment analysis.

This work builds directly on previous work on tagging and lemmatization contextualized embeddings (Straka et al., 2019a), uses existing datasets for all tasks, and aims to reach new state-of-the-art (SOTA) results. In addition to achieving better results, the aim of this work is also to explore some training techniques for transfer learning and compare the results, especially the case of fine-tuning versus full training from the beginning. The last goal of this work is to produce a set of publicly available models for non-commercial purposes, public source code and an accompanying text, which can serve as an introduction into the problem and a basis for further experiments.

Text structure

The following text is divided into four chapters: First chapter presents the theoretical background in NLP and used Artificial Intelligence (AI) methods. This quite general chapter is followed by a description of all performed experiments, which is presenting introduction into experiments and thorough description of each implemented task: definition, previous work, state-of-the-art results, methods applied in this work, and their results for Lemmatization and part-of-speech tagging and Sentiment analysis. Implementation details like code overview, third-party libraries and informations for personal examination and exploration of presented models can be found in chapter 3. Text is closed by a conclusion with a summary of contributions and future work proposals.

1. Theory

This chapter is divided into three parts. The first part introduces basic concepts of linguistics and natural language processing. With the focus of this work in mind, deep learning basics are presented in the second part. The third part of this chapter offers a more detailed explanation of methods directly relevant to this work (especially the BERT model).

1.1 Linguistics and Natural Language Processing

Natural Language Processing (NLP) can be described as a science at the border of linguistics and computer science. However, according to (Wilks, 2005), NLP itself is not a scientific research subject. It is instead a collection of problems, which can be examined. These tasks are taken from the general linguistics field, and the goal is to solve (or *process*) them by computers. The study subject of linguistics is language and its description. The focus of linguistics can be divided into the following sub-fields: phonetics, phonology, morphology, syntax, semantics, and pragmatics. This work focuses on morphology (lemmatization, part-of-speech (POS) tagging) and semantics (sentiment analysis) tasks. A more detailed description of tasks can be found in the following subsections and dedicated chapters for each task. Apart from the introduction of tasks, this chapter also includes a brief NLP history overview and a presentation of possible data sources for NLP training.

1.1.1 Morphology

Morphology studies an internal structure of words. Morphological tasks can be divided into generative and analytical. Generative tasks for a given word focus on the generation of word form for a given grammatical category. On the contrary, analytical tasks try to find e.g. a part of speech tag or grammatical categories of the given word. Both of these types of tasks are important for NLP.

This work performs two morphological analytical tasks – lemmatization and part-of-speech (POS) tagging.

Lemmatization Task

Lemmatization task consist of finding a *lemma*. Lemma is one chosen form of a word, selected to represent the whole set of all possible word's forms (such set is called a lexeme). A convention chooses word form used as lemma – it is nominative of singular for a noun, an infinitive for a verb, etc.

For example, lexeme for a Czech word *jablko* is *jablko, jablka, jablku, jablkem, jablek, jablky, jablkům, jablkách* and the lemma is *jablko*.

POS Tagging

POS tagging classifies word into one of the POS categories (like a noun, pronoun, or verb) (Hladká, 1998). A determination of grammatical categories (e.g., case, number, or tense) is sometimes also considered as a part of tagging task.

1.1.2 Semantics

Semantics deals with word meaning. This is a more challenging study than morphology, even for humans, let alone for computers. The most important reasons are that word meaning can be subjective, change during historical periods, a sentence is not a simple sum of meanings of its words, and moreover it is not clear how to represent meaning in computers.

Semantic analysis can be useful for various tasks, from natural language text generation to recognizing homonymy[1] or polysemy[2] of given words. It could solve sophisticated assignments as answering questions about the input text document, or help in high-quality translation, finding so-called named entities (like persons, months, or cites), linking these entities to some knowledge base, or analysing sentiment (which is one of the tasks solved in this thesis).

Sentiment Analysis

An input of sentiment analysis is a text, and the output is a classification into one of the categories. In this work, categories are positive, negative, and in some datasets also neutral, but it is common to use labels like abusive or ironic, too. As a part of sentiment analysis, the so-called subjectivity can be involved (Montoyo et al., 2012). The subjectivity prediction goal is to classify if the opinion (both positive or negative) is objective or the author is personally interested, and has strong emotions about his claims. For example, the following text could be recognized as objective: "The sound of this notebook is clear.", "The base is not stable enough." or even "An internet connection in this area is bad." in contrast with "I hate the way the new touchpad works.", which is highly subjective. The subjectivity of the claim does not depend on its sentiment. This work treat only positive/neutral/negative classification, not a subjectivity analysis.

1.1.3 Language Data

Data of many kinds can serve as an input into natural language processing, and this data can be categorized by a form or by source. As for form, we can work with corpora or datasets of various sizes, containing data from many sources. A corpus is a large collection of texts, aiming to be a representative sample of a language. This is not entirely possible, mainly because the selection of examples in the corpus is limited compared to language diversity. Despite these limitations, corpora are a valuable source of language information and are widely used in NLP. The most famous linguistic corpora include the Brown corpus (Francis and Kucera, 1979) – first electronic corpus mixed from newspaper articles and

[1] Homonyms are words, which share same spelling or pronunciation, but they have different meaning.

[2] Polyseme is a word with many different, but related meanings.

fiction literature, and PennTreebank (Marcus et al., 1993), which is the first syntactically annotated corpus, but has quite a domain-limited source – articles from the Wall Street Journal. Corpora can differ in internal structure. One of corpora's type is a treebank. A treebank is a corpus with many possible types of annotations that uses trees to represent dependencies. An example of such trees can be seen in figure 1.1 and an example of a treebank is presented in picture 1.2. A question arises as to what is the difference between a corpus and a "simple" dataset. Sometimes these terms can be interchangeable in the sense that the usage of both can be the same. Both types of data can be used for the same task, but the difference can be seen in a purpose of a collection. A corpus idea is to collect a somehow representative sample of a language with annotations on many levels, which allows the performance of various analysis upon this data. Dataset is typically created on a restricted domain, and they are annotated for one type of task, e.g., tweets on US Airlines pages[3] or movie reviews[4], which are examples of such sentiment analysis datasets.

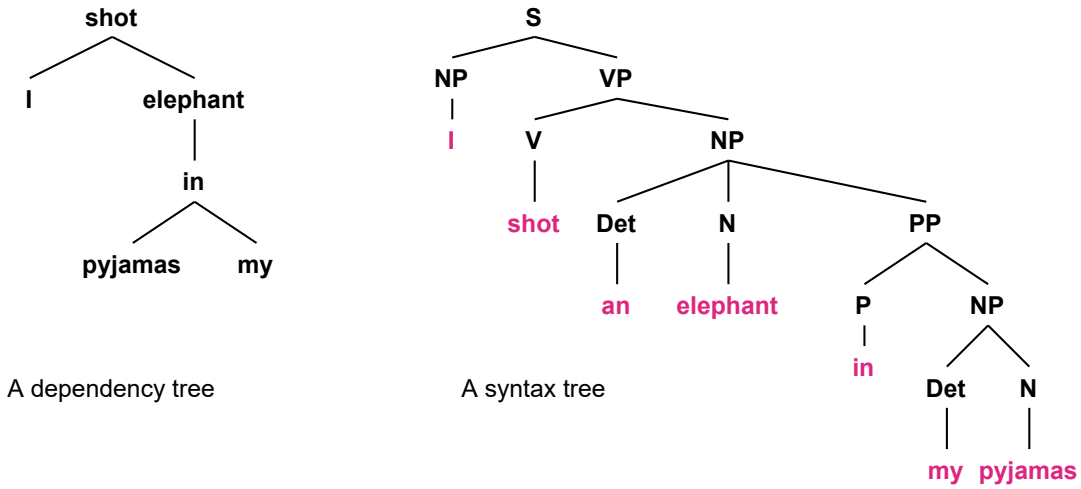


Figure 1.1: An example of the syntax and dependency tree. The dependency tree, as the name indicates, describes dependencies between words. Such dependencies are of various types; for example, an elephant in the example text is a direct object of the shooting action.
A root of such a tree is typically a predicate of the sentence. On the other hand, the syntax tree represents the sentence's syntactic structure according to the grammar. The root of the tree is *sentence*, which is split into noun and verb phrase. These can be further divided into phrases compound from particular instances of parts of speech (e.g., nouns, adverbs, verbs, prepositions, etc.).
Source: Bird et al. (2009)

Data in both dataset and corpora can come from many written or oral sources. For example, in machine translation task, documents with many language versions are appropriate. An example of the use of such multilingual documents was a project Eurotra (Oakley et al., 1995). Linguistic data can, however, differ

[3]https://www.kaggle.com/crowdflower/twitter-airline-sentiment
[4]https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

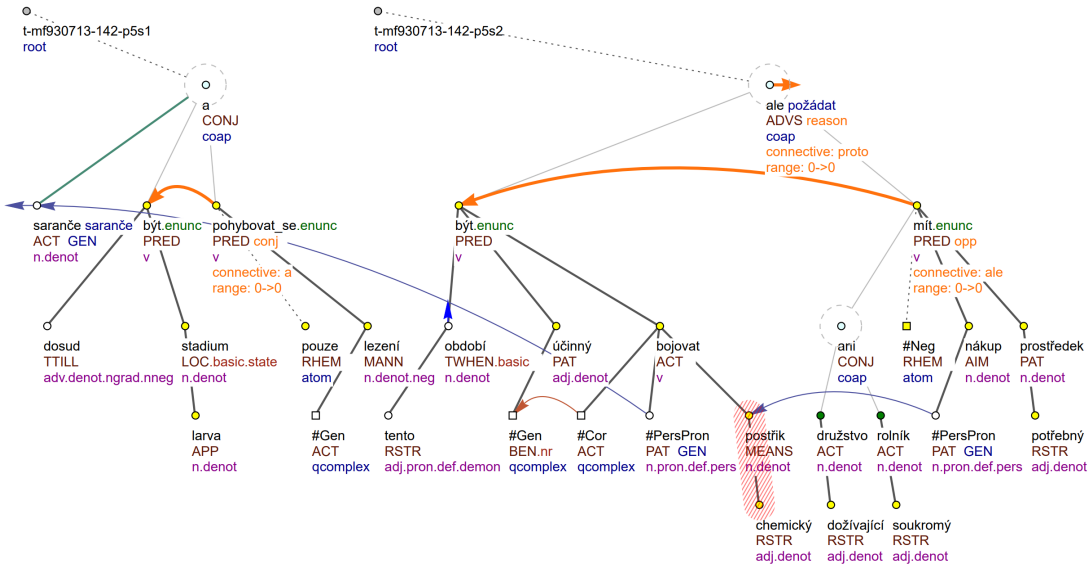


Figure 1.2: Prague dependency treebank example PDT35 for the sentences: *Grasshoppers are still in the larvae stadium, crawling only. At this time of the year, it is efficient to fight them using chemicals, but neither the ailing cooperatives nor private farmers can afford them.* Czech: *Sarančata jsou doposud ve stadiu larev a pohybují se pouze lezením. V tomto období je účinné bojovat proti nim chemickými postřiky, ale dožívající družstva ani soukromí rolníci nemají na jejich nákup potřebné prostředky.* This treebank contains dependency trees, but is is just one of many possibilities. Prague dependency treebank offers different layers of annotations. Red strips over words *chemický* and *postřik* marks multiword phrase, conjunction between *rolník* and *družstvo* is expressed as by one type of nodes, blue lines denotes coreference etc.
Praque dependency treebank data are used for tagging and lemmatization tasks.

in quality and length. A relatively new source of data are social networks like Twitter, Reddit, or Facebook. Data from some social networks (Facebook, Twitter) are very different from traditional sources like scientific papers, newspaper articles, or books. These data are short snippets of text full of odd characters, newlines, and ends of lines. They contain pictures, emojis, a mixture of different languages, slang expressions, and grammatical errors. Furthermore, they are very short; sometimes, they consist only of one sentence, few hashtags, and a link or a picture. Because people share their opinions and emotions and are ready to make their choices according to incoming influences, social networks are for some areas one of the most important source of information. The big problem while analyzing this data is their amount. Twitter users, for example, produce about 12 TB of data per day.[5] It is impossible to process all the data manually, so this is one reason for the rising industry importance of natural language processing.

1.1.4 Historical Development

The historical development of computer linguistics was significantly affected by machine translation (Wilks, 2005). NLP was improved by some of the milestones in machine translation history, therefore its historical development will be presented as well. First machine translation attempts formally started in 1933 by patents for machine translations (mechanical multilingual dictionaries) (Hutchins) followed by a big boom of machine translation in the 50s and 60s and then continued by a slowdown after ALPAC report in 1966 (Hutchins, 1996).

The first solutions to machine translation tasks were based on bilingual dictionaries and sets of rules. This method translated an individual word or a small group of words with a subsequent improvement of syntax and morphology. The resulting translation was not good and required lots of human work of expert linguists. This approach was replaced around the year 1990 by a statistical translation (Brown et al., 1990). A statistical machine translation's central idea is a probability of a translated sentence, given the original sentence. This includes also a probability of resulting sentence in the target language. This probability distribution is called a language model, and although it is one of the most aged ideas in NLP, it is an integral part of current best NLP solutions. Words probabilities were originally computed using frequencies of words or sequences of n rows (so called n-grams) in large language corpus (Jurafsky and Manning, 2012) (for more information about probabilistic language models, see subsection 1.2.7). These methods were quite successful, but they suffered from the curse of dimensionality [6] (Goodfellow et al., 2016, p.450). The next stage of machine translation (and other NLP tasks) starts with the second wave of neural networks' popularity (Goldberg, 2015),(Google). There are too many possible words or n-grams of words, and there is no way to share learned information between similar words or sequences in statistical methods. A solution to this problem is an invention of word embeddings (Bengio et al., 2003) (see 1.2.3). Neural language models (a neural network which learns probabilities of words) obtained even better results

[5] https://bigdatashowcase.com/how-much-big-data-companies-make-on-internet/

[6] Curse of dimensionality is a problem connected to the data with many variables. In such high-dimensional space, distances between samples are so big that it makes really difficult to distinguish between similar and completely different samples or find some meaningful pattern in the data.

(Schwenk et al., 2006). Current natural language processing is built on deep recurrent neural networks and encoder-decoder architecture, firstly published in (Cho et al., 2014), (Sutskever et al., 2014) and (Wu et al., 2016).

1.2 Deep Learning

In current times, neural networks are applied to machine translation and almost any other linguistic tasks. State-of-the-art result for machine translation, sentiment analysis, and many others is held by methods based on deep learning.[7] Also the main focus of this work is on neural network methods. For that reason, this section presents deep learning basics, their usage, and improvements in NLP in recent years.

1.2.1 Deep Learning History

The history of learning algorithms inspired by a human brain started in the 1940s under the name *cybernetics* (Goodfellow et al., 2016; McCulloch et al., 1943). The first such architecture was a perceptron (Rosenblatt, 1958). Perceptron, invented in 1958, is the most straightforward neural network with just one layer serving for binary classification (see figure 1.3). A perceptron's input is a vector of features describing an input example, and output is a classification into class 0 or 1.

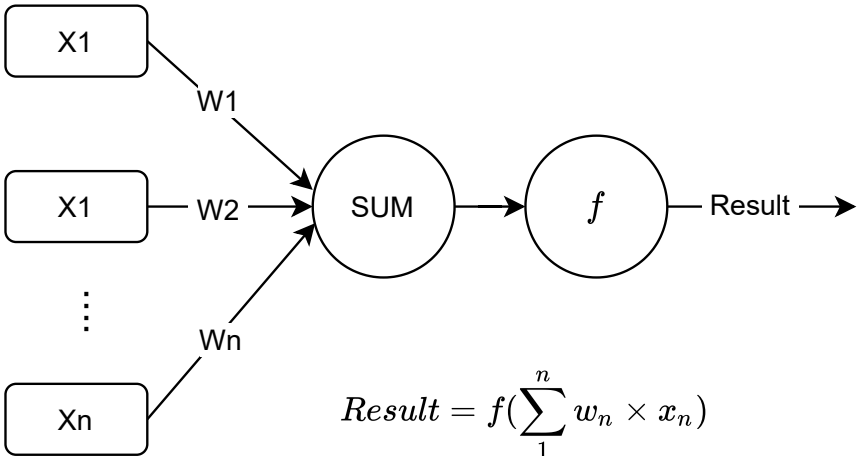


Figure 1.3: A one-layer perceptron architecture. The result is formed by the application of the activation function on a weighted sum of inputs. Weights are updated during training till it returns satisfactory results.

There also exists a multi-class version for general classification, but perceptron also had limitations. The problem of the perceptron was the inability to classify data that are not linearly separable (Minsky and Papert, 2017) (see figure 1.4), which led to a lack of interest in artificial neural networks for some period.

[7] `http://nlpprogress.com/`

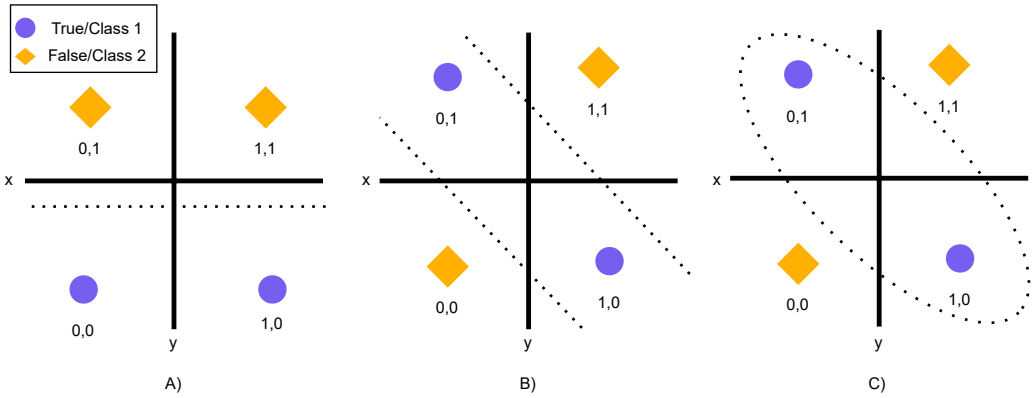


Figure 1.4: This picture illustrates the XOR problem. Perceptron can find the correct solution only if the data are linearly separable. It means that they can be divided by a hyperplane. An example of such two-dimensional data can be seen in picture A). The dotted line shows a possible border for separation. Picture B) shows XOR problem. XOR is a logical operation on two boolean variables, which returns true if one variable is True (1) and the other one is False (0), and returns False otherwise. Such data cannot be separated by one hyperplane. Linearly non-separable data can be, for example, separated by an ellipse (picture C).

The era of *deep* neural networks started around the year 2007 (Goodfellow et al., 2016) with bigger datasets and greater computational resources. These two new features opened the possibility of neural network learning without expertly handcrafted parameters tuning with good results. Many ideas, which are currently frequently used, are quite old - like backpropagation (Rumelhart et al., 1986) or even the encoder-decoder architecture (Allen, 1987), (Forcada and Ñeco, 1997). However, they became popular only after the development in other computer science areas (mainly because of more advanced hardware) reached a level where they can be trained in a reasonable time.

The basic type of a deep Neural Networks (NN) is a multilayer perceptron (see figure 1.5). It is composed of neurons; every neuron has an *activation function*, which is applied to its input. Input to every but the first layer is a weighted combination of (possibly selection of) neurons from a previous layer. Different NN types differ by the number and the shape of layers, activation functions, and connections between neurons (see figure 1.5).

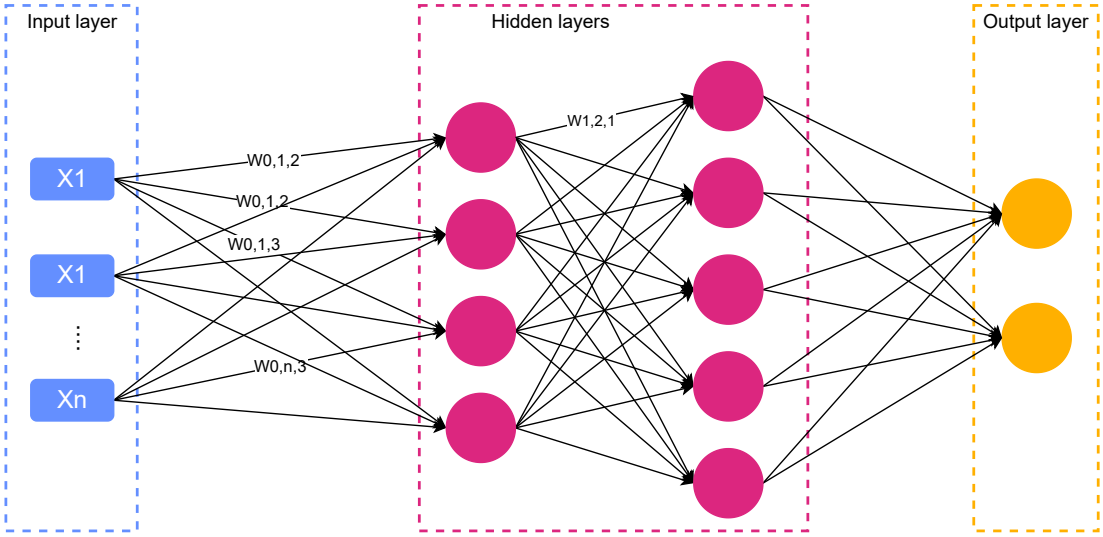


Figure 1.5: Multilayer perceptron (or feed-forward neural network) is formed of an input and an output layer and a variable number of hidden layers with different sizes. In every layer, the chosen activation function is applied to a weighted sum of inputs from the previous layer. In the illustration, the output is a probability for each of both classes in binary classification.

The following subsections present some of the key ideas for (not only) NLP and specifically for BERT. In NLP, the same as in other machine learning methods, it is necessary to decide how to encode the input to make it processable by computers, especially which input features are important for the given task and should be included. These questions are addressed in NLP, among others, by two techniques: word embeddings (see section 1.2.3) and an attention mechanism (see section 1.2.4). Word embeddings deal with the representation of words and their meaning, while the attention mechanism determines which parts of the text are relevant for a given task. Many machine learning methods are applied to data with no defined order between samples, like images or descriptions of petals for each sample flower.[8] Language data are, however, different, because their nature is sequential. Word and sentence ordering is an essential part of the text, and lack of it can make text absolutely senseless. This problem can be more or less satisfactorily handled by Recurrent Neural Networks (section 1.2.5) and Transformers architecture (section 1.2.6). Combining all these methods led to a BERT models family, which are used in this thesis (see section 1.3).

1.2.2 Machine Learning and Regularization

Machine learning is a computer science field dealing with algorithms, which can learn from experience and improve themselves. More precisely, "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell, 1997). For the techniques examined in this work, experiences are language data – every single experience is a text.

[8] `https://archive.ics.uci.edu/ml/datasets/iris`

To provide some measure, it is needed to know the ML algorithm's correct output. Many deep learning methods are constructed as *supervised*, which means that every experience has a corresponding label with a correct response (so-called gold data). Metrics used in this case reflect the portion of correctly predicted labels (Russell et al., 1995). Such data structure is handy for machine learning, but it is hard to obtain the data in the required amount because they are usually created manually by humans[9]. Opposite to this approach are *unsupervised* methods, where no labeled data exists, and metrics are based on different result features (e.g., the compactness of resulting groups). As for tasks, it is possible to distinguish them by the desired outcome between two basic categories: classification and regression. The classification consists of sorting data into one of the predefined classes (e.g., noun, adjective, verb); meanwhile, regression's goal is to predict a numerical result (e.g., expected number of borrowed books in a school library this year).

To be precise, supervised machine learning goal is not to predict all labels correctly in an example data (it would be enough to memorize them), but to predict correctly all possible inputs from the same distribution example data are taken from (find some general features for correct performance). During training, there can appear a problem called *overfitting*. As showed in Figure 1.6, overfitting problem is that the result prediction function practically memorized all training data examples and can minimize the error on them very nicely, but probably will not perform well on previously unseen data. A regularization is a tool for preventing such issues. Well known regularization techniques like lasso regression (Tibshirani, 1996) or ridge regression (Hoerl and Kennard, 1970), which are used for linear regression, work by adding some new members into the sum for the loss function, which should be minimized.

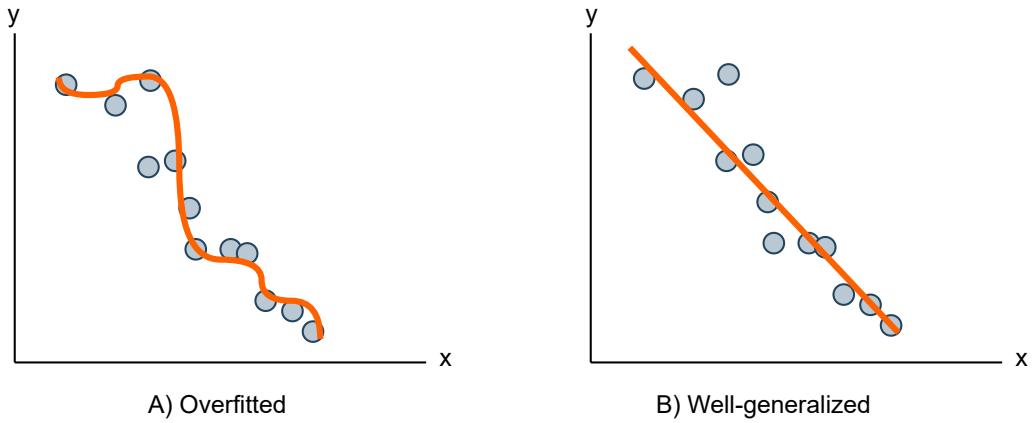


Figure 1.6: Figure A) presents overfitting scenario and Figure B) illustrates possible well-generalized solution for data with two variables.

Another classification regularization method, which is also used in this work, is label smoothing. Label smoothing (Szegedy et al., 2015) is an idea applicable to every classification problem, therefore is not limited to an NLP only. In any

[9]Obviously, if it were already possible to create labels by computers, there would not be necessary to learn it.

classification task, training data contains labels of the correct classes. In binary classification or one-hot encoded labels, correct class is denoted by 1 and incorrect class(es) by 0[10]. Instead of this, label smoothing applies following formula:

$$y_{new} = (1 - \epsilon) \cdot y + \epsilon/K,$$

where K is number of classes, y is the original label, y_{new} is the smoothed label and ϵ is the weight factor. Label smoothing is used as a cure for overfitting and overconfidence in the case of use of a softmax as output activation function. Loss function for softmax classification is:

$$loss = -\sum_{i=1}^{n}\sum_{y=1}^{K} p(y \mid x_i) log\, q_\theta(y_i \mid x_i),$$

with $p(y \mid x_i)$ being the truth labels' distribution and $q_\theta(y_i \mid x_i)$ being the predicted distribution of labels. After substitution of label smoothing:[11]

$$loss_{ls} = -\sum_{i=1}^{n}\sum_{y=1}^{K} [(1 - \epsilon)p(y \mid x_i) + \epsilon u(y \mid x_i)] log\, q_\theta(y \mid x_i),$$

for $u(y \mid x_i)$ uniform distribution (can be replaced by $1/K$) which gives after multiplication:[12]

$$loss_{ls} = \sum_{i=1}^{n} (1 - \epsilon)\left[-\sum_{y=1}^{K} p(y \mid x_i) log\, q\theta(y \mid x_i)] + \epsilon[-\sum_{y=1}^{K} u(y \mid x_i) log\, q\theta(y \mid x_i)\right] \tag{1.1}$$

From equation 1.1 can be seen, then if the network is very confidential about some prediction, the second part of the loss function is very large, so label smoothing works as a regularization of overconfidence.

1.2.3 Embeddings

Good performance of NLP models relies on a text representation. What is hypothetically desired is teaching computers to understand the semantics of the language. Once the computer has a good representation of what given text *means*, it should be easy to answer questions, translate it into another language, etc. Because NN can work only with a numerical representation of inputs, the second requirement upon such language representation is to be numerical. The straightforward way is to represent input words in one-hot encoding. In one-hot representation, a single word is represented by a vector, and one is only at the position of the respective word; all other positions are zeros. Such vector is long as a number of all possible distinct words, therefore it could be quite large. The most important problem of one-hot representation is that each two words are similarly distant from each other. It can be an advantage in some areas, but it is

[10] One-hot encoding transforms each label into a vector of size K, where n is a number of all possible classes. Than such vector is zero at all position expect the c-th position, where c is the correct class.

[11] Taken from: `https://leimao.github.io/blog/Label-Smoothing/`

[12] see footnote 11.

not a correct assumption in linguistics. Words can have similar meanings or be opposite to each other, generally spoken, a distance between them is not uniform.

When compared to one-hot encoding, embeddings (Bengio et al., 2003; Ling et al., 2016) are better solution for language data. Embedding is also a vector representing an input word, but in contrast to one-hot encoding, its size does not depend on the vocabulary size. These embeddings are learned by neural networks instead of being prepared by humans. They can be learned for every specific task from scratch, or it is possible to use embeddings trained for usage in many tasks like in the following cases.

Non-contextualized embeddings

Before contextualized embeddings appeared, pretrained embeddings were created mainly by Word2Vec (Mikolov et al., 2013), (Turian et al., 2010), (Pennington et al., 2014), specifically by its two variants: the CBOW and the SkipGram model. The objective of CBOW model is to predict a missing word from its context, and SkipGram does precisely the opposite – predicting the context of the given word. These predicting objectives serve just as a tool for forcing a network to learn a useful word representation. Embeddings are then input into a network through a layer with size $number\ of\ words \cdot embedding\ size$. We still need to bridge the gap between text and numbers, which is possible to do using one-hot encoding or simple word numbering as an input into this first embedding layer. The problem of Word2Vec-like embeddings is that the embeddings depend only on a few nearest words, but the statistics in the dataset are not explicitly used. GloVe (Pennington et al., 2014) embeddings, on the other hand, use information about frequencies of pairs of words in a whole dataset, and are designed to project word vectors into meaningful vector space.
Embeddings of previously described types use a context of the word – it is, in fact, the way how they are meant to work. Similar words are supposed to appear frequently in a similar context. The problem of such embeddings is that an input word embedding in the first layer of the neural network is computed independently on neighbor words, so the same word always has the same embedding regardless of the context. In the case of homonyms, non-contextualized embeddings are a mixture of all the (possibly very different) meanings, which can lead to poor results. The main problem for same word embedding in a different context are homonyms. To solve this problem, contextualized embeddings were developed providing better results universally (Straka et al., 2019b; Liu et al., 2020).

Contextualized embeddings

Contextualized embeddings were invented in recent years, namely ELMo (Peters et al., 2018), BERT (Devlin et al., 2019), and XLNet (Yang et al., 2019a). Their comparison can be found in a subsection 1.3.4. The main difference from non-contextualized embeddings is that same words obtain different meaning according to the sentence they are part of. In addition, they also take into account a larger context than the above-mentioned non-contextual methods. Section 1.2.7 describes the possibilities of involving and training such embeddings in more detail.

As embeddings are trained after the input is encoded into one-hot vectors, it is impossible to use pretrained embedding for the encoding of previously unseen words. This problem is solved by embeddings of characters or subwords, so that the whole word embedding can be later a compound of them. Embeddings are currently the best option of an input representation.

Another problem is recognizing which parts of the input are valid for the given task. This problem is addressed by the attention mechanism, described in the following subsection.

1.2.4 An Attention mechanism

Attention mechanism (Bahdanau et al., 2014) is widely used in NLP as a tool for extracting relevant information from word sequences. For example, when generating a sentence translation, each word in the target language corresponds to just a few words in the source sentence, not to a whole sentence. Attention gives weights to words, which represents this connections (see picture 1.7).

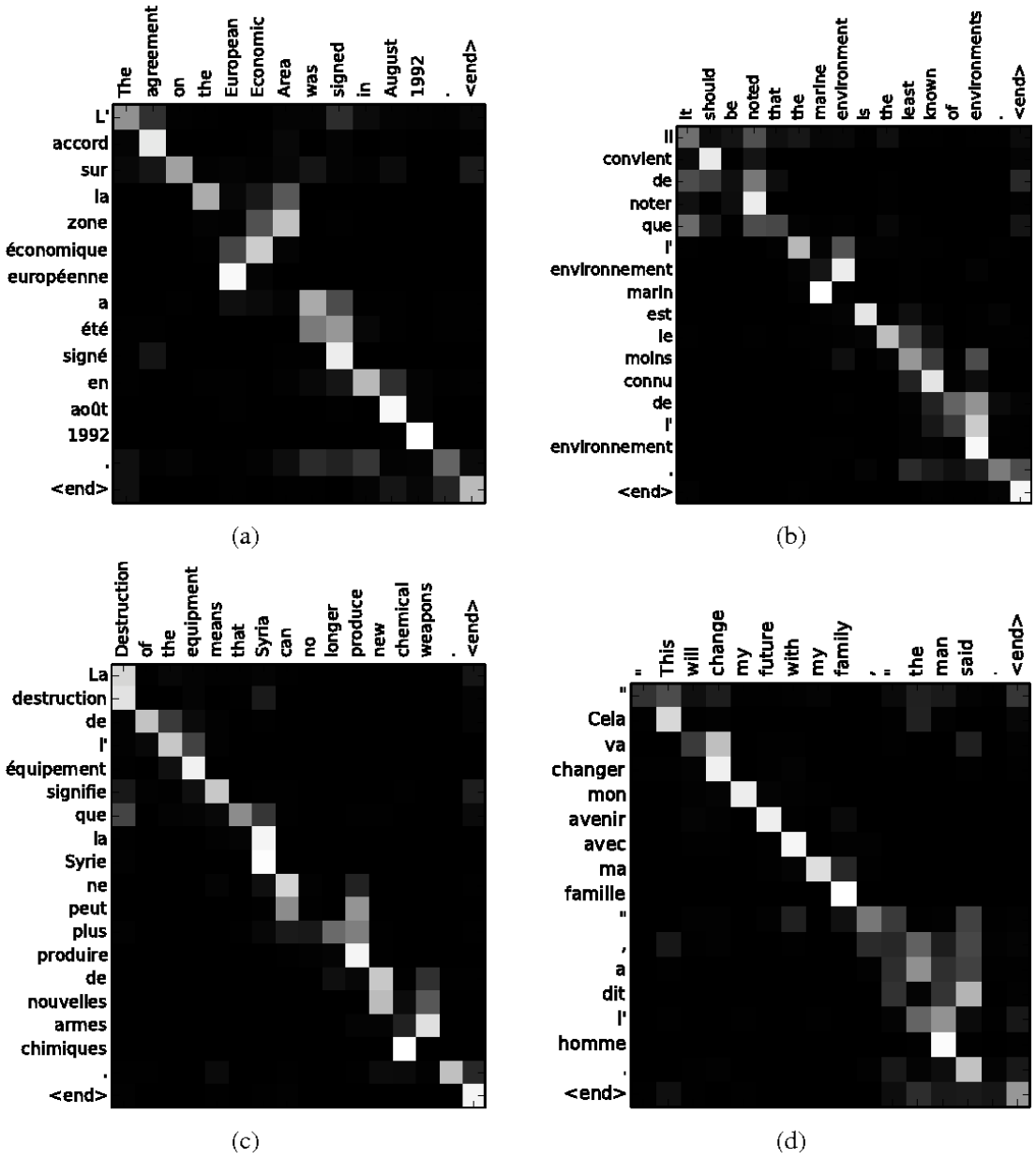


Figure 1.7: Figure 3 of (Bahdanau et al., 2014) presents the use of the attention for machine translation. Axis show words of a sentences in English and French. The importance for a translation between a pair of words is represented by a lightness (lighter = more important).

When the task is question answering, attention can help a model focus on a relevant part of the text, where the answer is located (dos Santos et al., 2016).

The same idea can be applied to computer vision, where it imitates human behavior. Humans also focus on (or *attend to*) just a few parts of their visual input when they are, for example, recognizing things in pictures. Modification to an attention concept, called self-attention (Cheng et al., 2016), deals with relationships inside one part of the text (e.g., a sentence). This variant of attention does not connect one part of the text (like a question) to another distinct part

of the text (like an answer), but only models relationships inside one part. For more explanation, see figure 1.8.

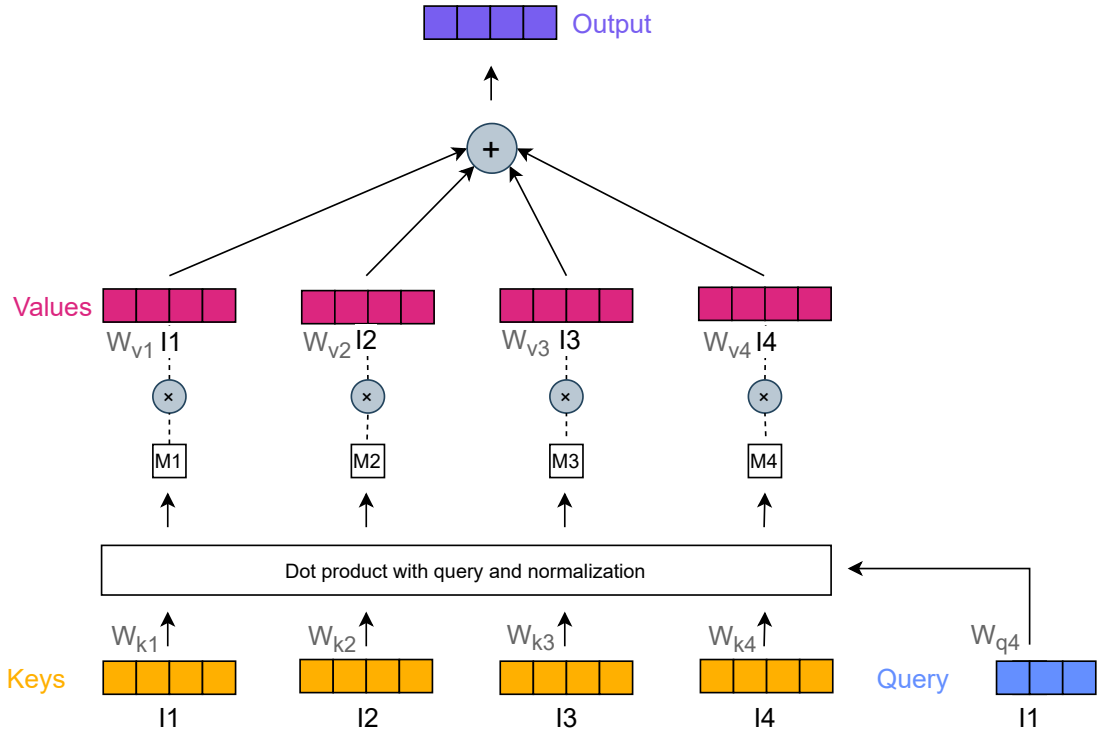


Figure 1.8: Self-attention mechanism scheme for one selected query vector. The result is an embedding, which is improved by the context of the word. This picture illustrates the result for the embedding of the first word (l1) in a four-word long text. Input words are denoted as l1 to l4. Keys, values, and a query are all computed by multiplying the input embeddings by their respective weights (W_k, W_v, and W_q) before any other operation with them. These weights are trained during learning. Dot products between every word and every query are computed. The result is a number for every input word, so four numbers at the end. These numbers are normalized, so the sum of them is equal to 1. These numbers serve as a weight (M_x), which indicates the relationship between the query and every other word. The resulting better embedding for the query is then obtained as a sum of the word embeddings weighted by these obtained weights.

An improvement to the self-attention – *multihead attention* (Vaswani et al., 2017) also tries to model relationships between the words in the same sequence. As it is multiheaded, it can, for one word, pay attention to more words (or their parts) (see figure 1.9).

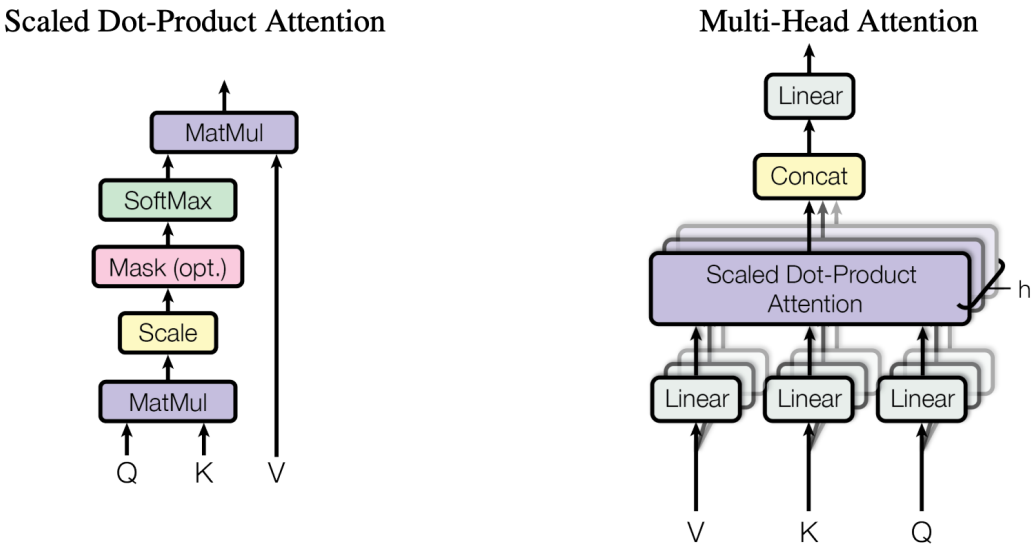


Figure 1.9: Figure 2 from (Vaswani et al., 2017) describes the attention mechanism used in Transformers. Scaled Dot-Product Attention, as authors call it, is basically the same architecture as described in the previous figure (1.8). The dimension of values (d_v) is different from query and keys dimensions (d_k). As the normalization serves scaling (division by the number of input dimensions) and than softmax function, which ensures the sum of all weights to be equal to one. Multi-head attention just perform Scaled Dot-Product Attention in parallel and result is then concatenated.

1.2.5 Recurrent Neural Networks

Text sequences can be very long, and related words often have a long distance in between. This fact places challenging demands on neural networks because such data structure differs from most other NN applications, where input samples are independent, and order does not matter. Text size can also lead to vanishing/exploding gradients because information should be carried for many steps, leading to many multiplications of very small or big numbers in neural networks.

Based on the above mentioned, the construction of neural networks than can capture natural language structure requires solving two issues:

- input should be understood by the network as a sequence,
- there must be a possibility to use information from other parts of the sentence (and not to forget them).

The first problem was solved by simple Recurrent Neural Networks (RNN). To represent an ordering and a continuity of input words, basic RNN takes the output for previous word as a part of input for the next word (see figure 1.10).

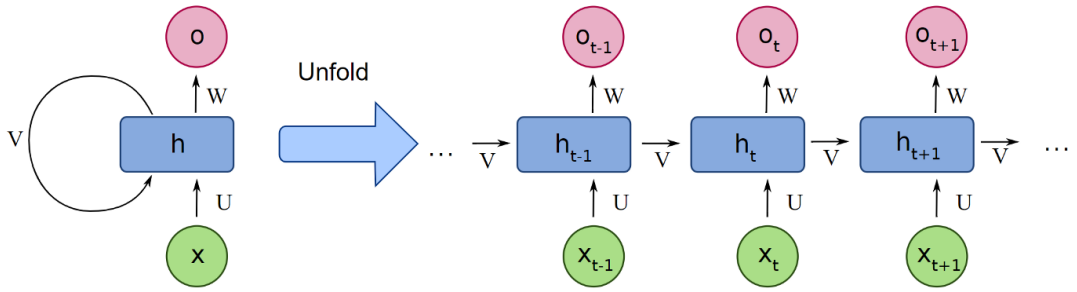


Figure 1.10: Basic Recurrent neural network architecture is showed on the left side of the figure. It is composed by one rnn cell which recurrently uses informations from previously seen input. X is the input of the cell, e.g. a word, O is the output for the given word (e.g. its translation or next word prediction), V is the state passed into another time step. For better illustration of working in the time, RNN can be visualised as a chain of cells connected by a result of previous cell as can be seen on the left side of the figure. Source: *Picture from* `https://medium.com/deeplearningbrasilia/deep-learning-recurrent-neural-networks-f9482a24d010`.

The latter problem needs a more complicated approach. There are three attempts to solve this *short memory* of RNN cells – Gated Recurrent Unit (GRU) (Cho et al., 2014), Long Short Term Memory (LSTM) (Hochreiter and Urgen Schmidhuber, 1997) and Transformers architecture (Vaswani et al., 2017) with an attention mechanism.

Both LSTM and GRU uses an idea of gating. Term *gate* refers to a weight (multiplication factor) for previous informations and new input. Gate determines which information from previous words should be remembered and which should be forgotten. For regulating the memory, the gate is formed by the sigmoid activation function, ranging from 0 to 1 and presenting a portion of remembered information. Previous information is encoded in a cell state and a hidden state, both passed from cell to cell (with the application of the gates). Comparison of both architectures can be found in figure 1.11.

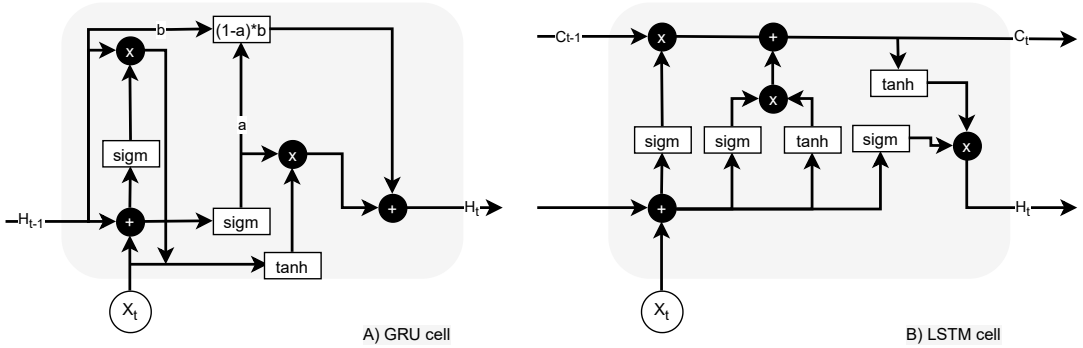


Figure 1.11: Comparison of LSTM and GRU architecture. x_t is the current input (current word), Source: `http://dprogrammer.org/rnn-lstm-gru`.

LSTM

LSTM cell uses three gates: input, output and forget gate. Every gate is composed by a sigmoid function with an actual input and a previous hidden state as inputs. *Forget gate* filters information from previous cell state. *Input gate* decides which parts of input will affect the results. *Output gate* then selects which part of the result will be actually part of a result. Detailed description can be seen on Figure 1.12.

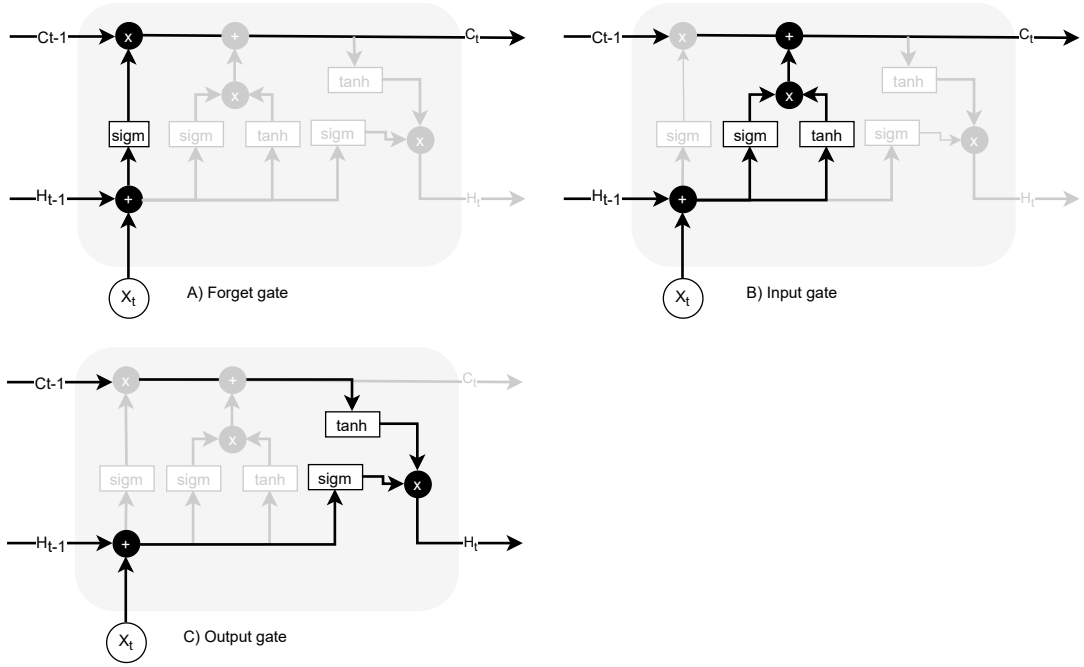


Figure 1.12: An architecture of the LSTM cell, epsecially its gating mechanism. A LSTM cell carry through the time, in addition to a hidden state, also a cell state, which serves as a long-term memory. *A forget gate* is responsible for choosing the amount of cell state, therefore the information from previous inputs, to be preserved. It uses sigmoid function which returns the output between 0 and 1. *An input gate* controls the addition of new information from the input to the memory. *An output gate* produces the output hidden state which is passed to the next cell.

GRU

GRU also uses gates: reset gate and update gate. The *reset gate* is responsible for how much of the previous state will take in the new state. *The update gate* serves as a weight for a combination of previous and current states, which form the new output. For more details see Figure 1.13.

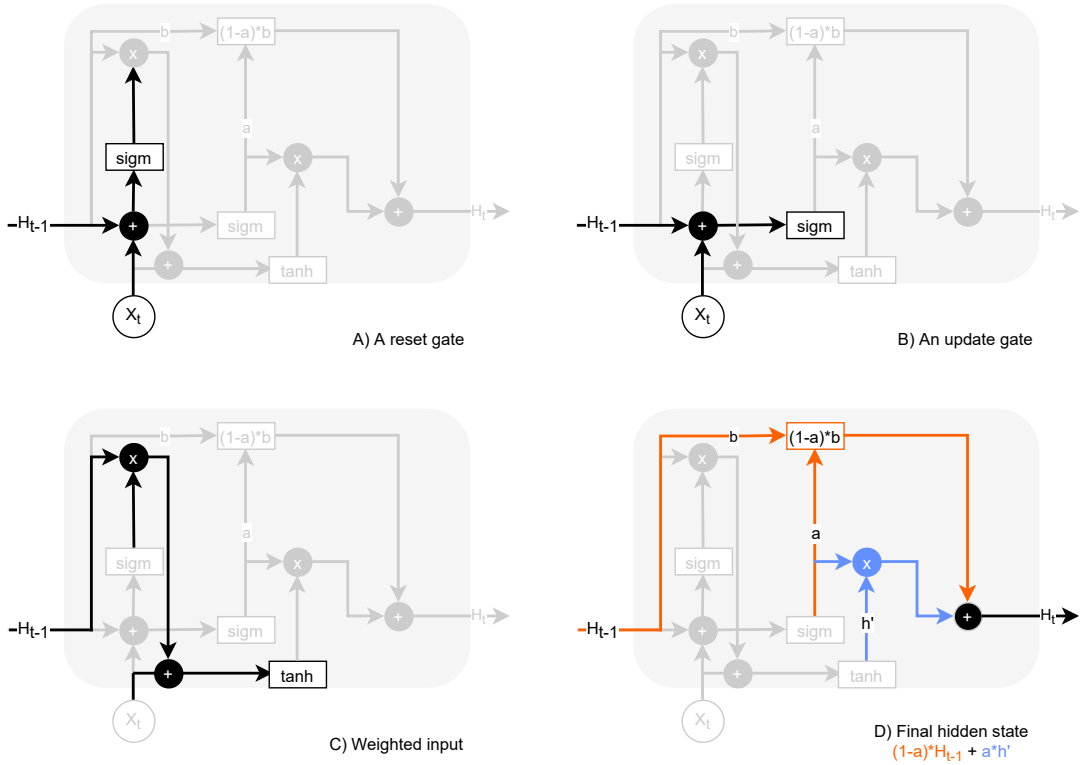


Figure 1.13: An architecture of the GRU cell. *An update gate (A)* computes how much of previous information should be passed to next cell based on previous hidden state and the current time input. *A reset gate (B)* uses the same formula (although the input and the previous hidden state have different weights) but serves to a different purpose – it decides which to forget. *(C)* previous hidden state is weighted by the reset gate before concatenating with current input and normalized via tanh function (to be between -1 and 1) Finally, *new hidden state* is computed as an affine combination of previous hidden state and "normalized" hidden state from previous step.

1.2.6 Transformers

The Transformers solve the same problem as RNNs,s but propose a different architecture. The Tranformers architecture was proposed in 2017, in a paper Attention Is All You Need (Vaswani et al., 2017), and essentially depends on a self-attention mechanism (see subsection 1.2.4). Tranformers uses encoder-decoder architecture, which was simultaneously published in 2014 by (Cho et al., 2014), (Sutskever et al., 2014) and (Wu et al., 2016). This architecture serves to processing of variable-length sequences. Encoder and decoder are connected by a vector of fixed size (context vector), which aims to be a good representation of the input. The encoder reads its input and tries to learn such weights that the encoder's final representation of the input contains all important information. This context vector serves as an input into the decoder, which tries to reconstruct the best results. This architecture was first used for machine translation, so the decoder's output, in this case, is a sentence in the target language with the same meaning as an original input (see figures 1.14 and 1.15).

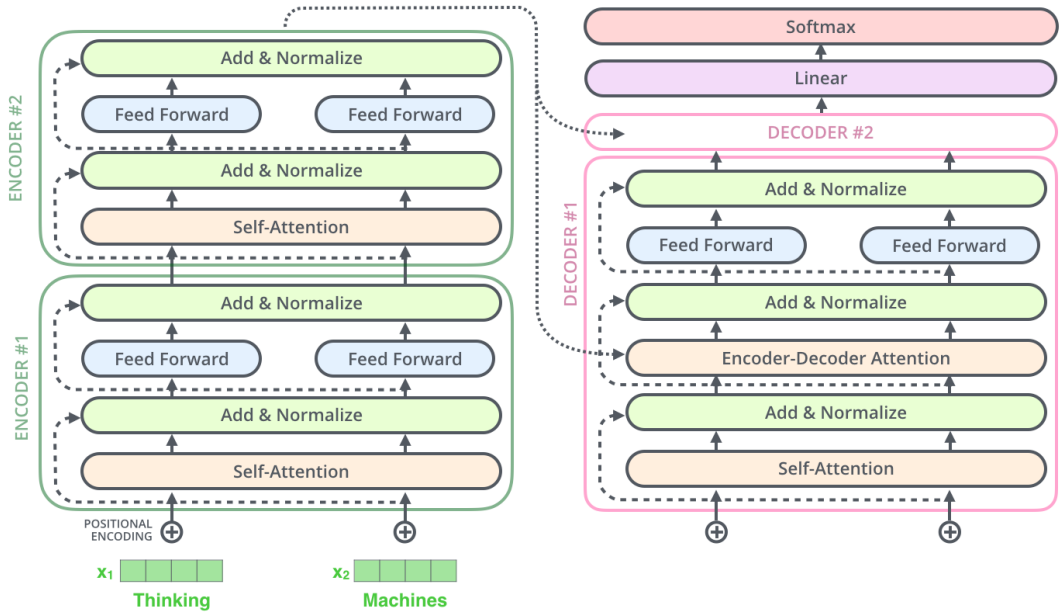


Figure 1.14: This picture describes design of one encoder-decoder block in detail. Every encoder layer consist of self-attention and feed-forward layer supplied with normalization and residual connections. Source: `http://jalammar.github.io/illustrated-transformer`

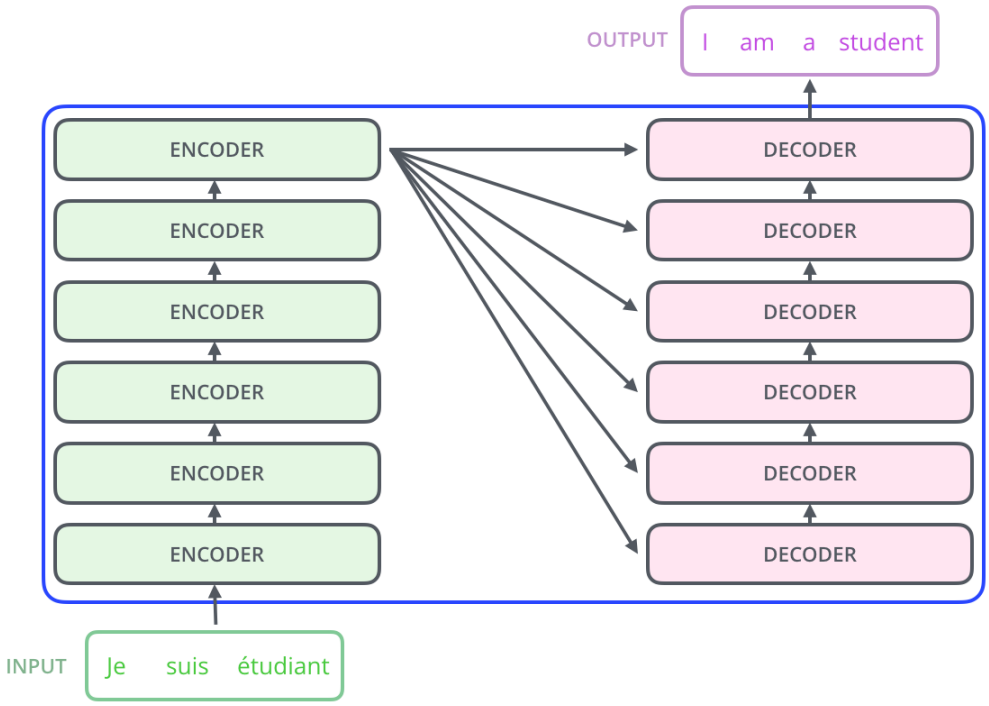


Figure 1.15: In transformers, encoder and decoder parts are both composed by many of block of respective types. The input goes first through a series of encoders and than the output of encoder part is put into every decoder in the decoder part. source: `http://jalammar.github.io/illustrated-transformer/`

The self-attention mechanism is supposed to select the most important words to be focused on and used in many places – for the input of every encoder layer, for the input of every decoder layer (although masked), and also between encoder and decoder. On the decoder side, the self-attention layer is masked so the decoder can "see" just the previous words (there are $-\infty$ values in the positions on the right or the current positions). For a representation of word position in a sentence (as it is not an RNN cell and it can process all words simultaneously), Transformers use position embeddings, which are trained to represent the sentence's ordering.

1.2.7 Transfer learning

Transfer learning is a very important idea because it allows, as the name suggests, to transfer learned knowledge between different tasks. Reusing the knowledge can lead to lower training times with fewer demands on technical resources (GPU, CPU) and training data size. It even allows to successfully apply automatic processing into domains where labeled data are not available by transferring the knowledge from another domain with enough training examples. In addition, usage of underlying common knowledge between different tasks can also improve the results of learning algorithms on each task. One of the first big successes of transfer learning comes from the computer vision field by using models pretrained on ImageNet (currently also on other datasets). ImageNet (Russakovsky et al., 2015) is a big dataset of pictures. Every picture is labeled by one of a thousand classes. Many large deep models were trained on this dataset and then applied to different computer vision tasks with a great success (Huh et al., 2016).

Following the taxonomy in figure 1.16, currently most important transfer learning applications in NLP falls into *sequential transfer learning* category. Machine learning models are first trained on a training objective, and the trained result is then used for a wider set of tasks.

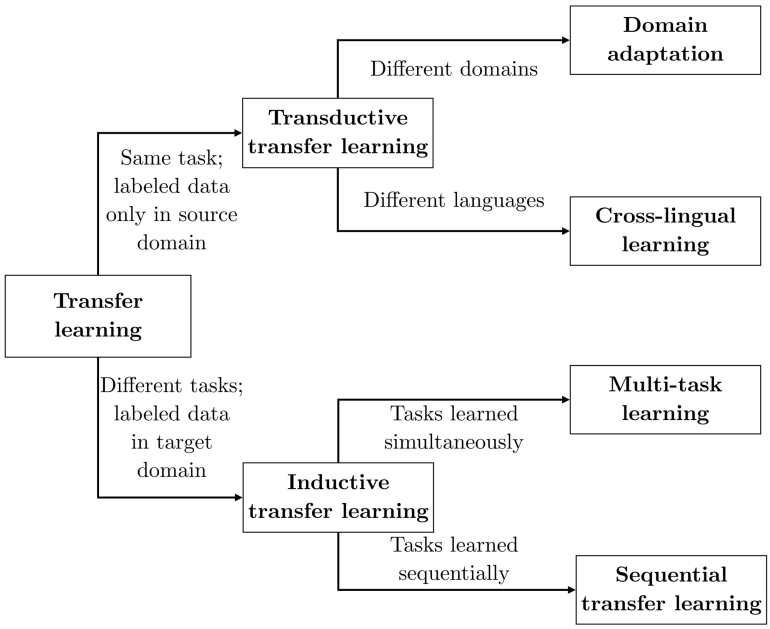


Figure 1.16: Figure from (Ruder et al., 2019) offers possible taxonomy for transfer learning. Following definiton in (Pan and Yang, 2010), transfer learning's goal is to improve the performance on task T_1 from domain D_1 by learning knowledge on task T_0 from domain D_0. Domain is defined as $D = \chi, P(X)$, where $X \in \chi$, χ is a feature space and $P(X)$ is a marginal probability distribution over the feature space. Transfer learning allows the use of trained models on tasks with different sets of labels or different input data's nature. Input data can vary in the source they come from (wikipedia text versus a novel or a social network posts), they can learn from different features (e.g. different languages) or the distribution of classes is different than it was in the training data (so some highly presented classes in training data are rare in this new task and others are quite common but previously not seen too many times).

For natural language processing, transfer learning is currently mainly represented by contextualized embeddings obtained from pretrained language models. The embeddings are not important only because it solves the problem of homonyms, but also because they are believed to store knowledge independent of any language (Feijo and Moreira, 2020; Hewitt and Liang, 2020). Contextualized embeddings can be obtained by both supervised or unsupervised learning (Liu et al., 2020). This work focuses on unsupervised learning, as it is currently a promising field according to recent results, and because unsupervised learning does not depend on large manually created datasets.[13] Supervised methods use machine translation, which is the classic NLP task, but also on natural language inference or other tasks with the potential to capture general knowledge about language.

[13]This is a slight terminological inaccuracy. In section 1.2.2, unsupervised learning was defined as a task, where data does not contain the correct answer. BERT and derived models, however, use pre-training tasks, where the input does not need the manual annotation, but in training itself, individual experiences are provided with the correct answer. Better term introduced by Yann LeCunn `https://twitter.com/ylecun/status/1123235709802905600?lang=cs` is *self-supervised*.

Unsupervised learning tries to learn a language model – a probability distribution over a sequence of tokens given by the following equation:

$$p(t_1, t_2, ..., t_N) = \prod_{i=1}^{N} p(t_i | t_1, t_2, ..., t_{i-1})$$

(Liu et al., 2020), where $(t_1, t_2, \ldots, t_N)$ is a sequence of tokens. To reduce this problem, it is possible to consider only fixed size sequences of $n - 1$ previous words for every word probability called *n-grams* (Bengio et al., 2003). To achieve the goal of learning a language model, one can use many different tasks, which are believed to force the network to learn useful knowledge about language. First attempts were made with autoencoders (read the text, encode it and try to decode it back) (Dai and Le, 2015) and machine translation (Ramachandran et al., 2017), but later papers come with better architectures and various new objectives, which are described later in section 1.3. There are generally two ways to transfer the learned knowledge (Feijo and Moreira, 2020): extract some representation from the model and use it in another model without changes, or modify a model by changing the task-specific layers (so-called head) and *fine-tune* the whole newly created model for a specific task. There is also a possibility to combine both approaches and, at first, take static features as an input for training, and then when the head starts to perform well, fine-tune the whole model with this better head, so the original weights converge more efficiently to the wanted solution. More detail are offered in section 1.3.3.

1.3 BERT and its descendants

The methods described in previous section are utilized in the BERT-like models, which belong into a family of contextualized embeddings together with e.g., Contextualize Word Vectors (CoVe) (McCann et al., 2017; Peters et al., 2017), ELMo (Peters et al., 2018), Flair (Akbik et al., 2018), and series of Generative Pretrained Transformers (GPT) models (Radfort et al., 2018; Radford Alec et al., 2019; Brown et al., 2020). These models are important steps in NLP progression, which led to BERT family of models. BERT, representing a very effective contextual embeddings, demonstrated better ability to capture language knowledge and constitute an important milestone in the NLP.

First attempts to contextual embeddings appeared with two models: the first (CoVe; (McCann et al., 2017)) uses supervised machine translation and the second (Peters et al., 2017) uses unsupervised language modeling. Both these models are used for extracting embeddings. These embeddings are concatenated to the non-contextual embeddings (i.e., GloVe) for the target NLP tasks. CoVe uses a machine translation task (it needs a parallel bilingual dataset) and biLSTM encoder-decoder architecture. The biLSTM are LSTMs, which process text in both directions. CoVe, therefore, uses supervised learning, in contrast to the following presented models, which took the path of unsupervised learning, as learning from the raw text has a considerable potential due to easy access to a large amount of unlabelled text data (in opposite to labelled datasets, where there is almost always not enough data). Peters et al. (2017) use an unsupervised

method – language modeling and a concatenation of forward and backward RNN (similarly to CoVe, but uses both GRU and LSTM depending on the task). Both models use last layer as an embedding representation.

ELMo is third in the series of biLSTM architecture models and builds on Peters et al. (2017), but it uses a deeper representation of words. Embeddings are created from a weighted combination of all network layers (in the original ELMo, there are only two layers). This deeper combination was led by the assumption that different network layers are capturing different (but valuable) knowledge. Experiments with weights showed that lower layers tend to capture syntactic information and therefore are more important for syntactic task, while higher layers are important for semantic tasks.

Flair also uses unsupervised LM, but the smallest unit of the input is an character, not a word. Flair models the n-th character's probability given the previous characters in the probabilistic description of language modeling. An output is again a word embedding, but this time combined from a representation of its characters. The authors chose this approach to eliminate problems with unknown words.

GPT by OpenAI (actually in version 3) also uses language modeling for pre-training, but the difference is this time in the architecture. Instead of the LSTM-based RNN network, GPT variants use the decoder part of the Transformer architecture and the attention mechanism. GPT also presents deeper architecture than all previously presented methods. GPT 2 version proposes 4 model sizes, with the smallest one having 12 layers and the deepest one 48 layers. Each layer is a Transformer decoder with self-attention as described in section 1.2.4. GTP is deeper than ELMo and others, but only considers a left context of the word, as the text is processed sequentially in only one direction.

1.3.1 BERT

On top of all previously mentioned contextual embedding models stands the Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019) with the depth comparable to GPT 2, but using a different part of a Transformer architecture. It uses modified pre-training tasks compared to other classical language modeling and has a context from both sides. BERT is a pre-trained language model that is fine-tuned for many other tasks, so it is an example of the transfer learning (see section 1.2.7). To train a language model, original BERT is trained on two tasks – next sentence prediction and masked language modeling. BERT is proceeding both sides context simultaneously, and due to architecture, in *every layer* every Transformer block potentially has information from every other block (and thus from every other word). That is why BERT is called **deeply** bidirectional or non-directional (because there is no right-to-left or left-to-right direction of processing). BERT is very successful in solving NLP tasks, although it was surpassed in many tasks by later derived models. The second part of its popularity is the transfer learning feature of BERT. Resulting BERT model can be used for almost every NLP task just by changing the classification head. The

authors published pre-trained models for English and later also for Chinese and multilingual model and other monolingual models were also published by other authors.[14] Training of the language model requires a considerable amount of data and computational resources, but it is needed to be done just once. When the model is trained, it can be used for many tasks by changing the classification head and training only newly added layers or training all layers (but only for a few episodes and with smaller data) and still performing on a state-of-the-art level.

The core of BERT algorithm is based on these three features

- two unsupervised objectives for pre-training,
- input embeddings,
- encoder part of the Transformers architecture,

and all of them will be described in the following sections.

Input embeddings

BERT uses the concatenation of three types of embeddings as an input representation – token embeddings, position embeddings, and segment embeddings.

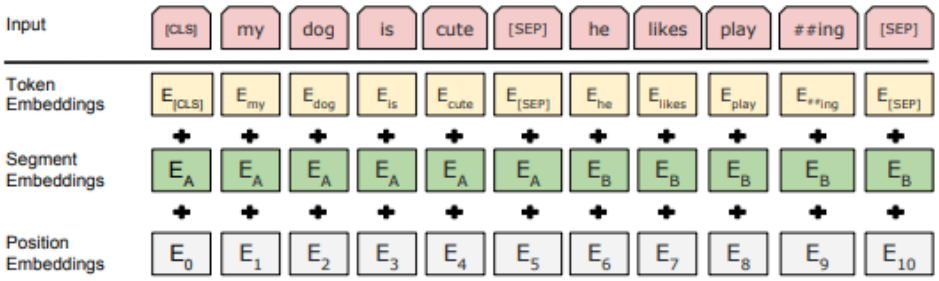


Figure 2: BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Figure 1.17: An input is represented using three kinds of embeddings for every input word. Every sequence is also marked off by beginning and ending markers (CLS and SEP token), which are also encoded using a combination of all three embedding stypes. Source: *(Devlin et al., 2019)*.

Token embeddings The BERT model input can be one or two sequences (not necessarily two sentences, but, e.g., also paragraphs).[15] All words are split into tokens and converted into embeddings with the use of a pre-trained embeddings model. One word can be tokenized into more tokens because BERT uses WordPiece embeddings (Wu et al., 2016). WordPiece pre-trained embedding algorithm was originally created for the task of Google voice search for Asian languages,

[14]Some published models implemented in the popular Python library Tranformers from HuggingFace, can be found here: `https://huggingface.co/models`.

[15]Different terminology is used here than in the original paper. In (Devlin et al., 2019), *sentence* is a term for a whole part of input (fist or second), while a term *sequence* is used for whole BERT input compound of one or two *sentences*.

and is designed to minimize the number of word tokens. WordPiece model was not pre-trained as a part of BERT paper experiments, but represented a quite interesting solution, so that the idea will be briefly explained here.

It is impossible to prepare embeddings for every possible word in a language, because this would cause an intractably long embedding size. Every word, which is not a part of the selected embedding set, is encoded in the same way (as an unknown word). This situation is not desired because we lose information about words. WordPiece deals with this problem in the following manner: In the first iteration of training, the model creates embeddings only for characters. In every other iteration, some existing model words are concatenated together in a way that causes the highest likelihood of the input text. As a result of this method, some words will be embedded as one word, and some will be split into more tokens, as can be seen in figure 1.18.

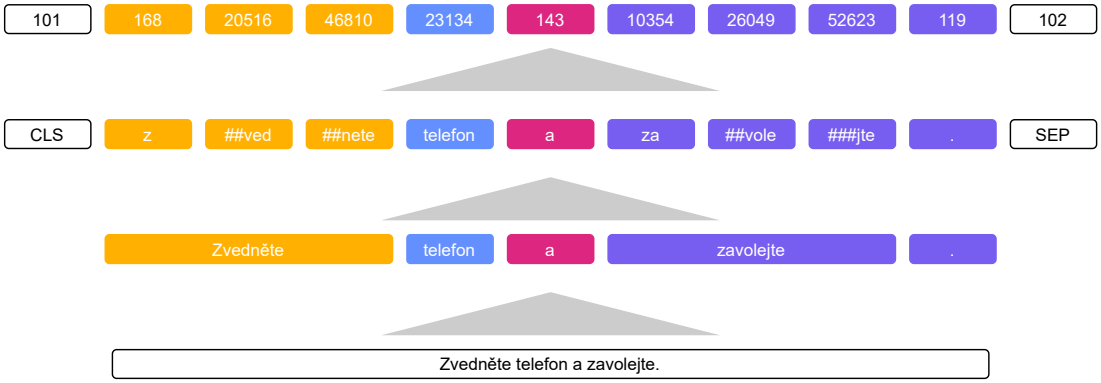


Figure 1.18: This figure illustrates a transformation of one input sentence (from PDT3) to suit BERT input expectations. The sentence is divided into words and then into tokens from WordpPiece tokenizer vocabulary. Accents may be removed depending on the used model. The sentence is decorated with special CLS and SEP tokens to mark the beginning and the end of the sentence. All tokens are then converted into numbers.

Three other tokens are added after this step – CLS and SEP. CLS token is added at the beginning of the input and is used as the first sequence embedding for classification tasks (as sentence analysis). SEP token separates both sequences and is also appended at the end. Whole input transformation can be seen on figure 1.17.

Position embeddings All input tokens are processed simultaneously. That is the reason why BERT is often called *undirectional* rather than *bidirectional*. This feature causes an absence of information about the order of tokens. However, the nature of the language is sequential. A bunch of words without an order has no language meaning, and capturing this problem led to recurrent neural networks at the first place. In BERT there is no recurrent cell, so instead, position embeddings are used to solve this problem. They have the same shape as token embeddings (and as segment embeddings), and they are learned the same way as other embedding layers. The original BERT's maximum input size is 512, so this embedding layer should represent positions from 1 to 512. This learning is

	BERT Base	BERT Large
L	12	24
H	768	1024
A	12	16
Total Parameters	110M	340M

Table 1.1: Difference between base and large version of BERT model, as published in (Devlin et al., 2019).

different from the original Transformer architecture, where the position was also encoded as embeddings, but the embeddings were fixed, not learned.

Segment embeddings These embeddings indicate whether a token belongs to the first or second part of the input. They have the same shape as position and token embeddings, and they are also learned. Because BERT input can consist of at most two parts, segment embeddings encode whether the token belongs to the first or the second part.

Architecture

BERT adapts encoder part of the architecture from the original Transformers paper (Vaswani et al., 2017) (see section 1.2.6). BERT uses its encoder architecture for each layer, so L encoder layers are followed with one fully connected layer for a specific task (see figure 1.19).

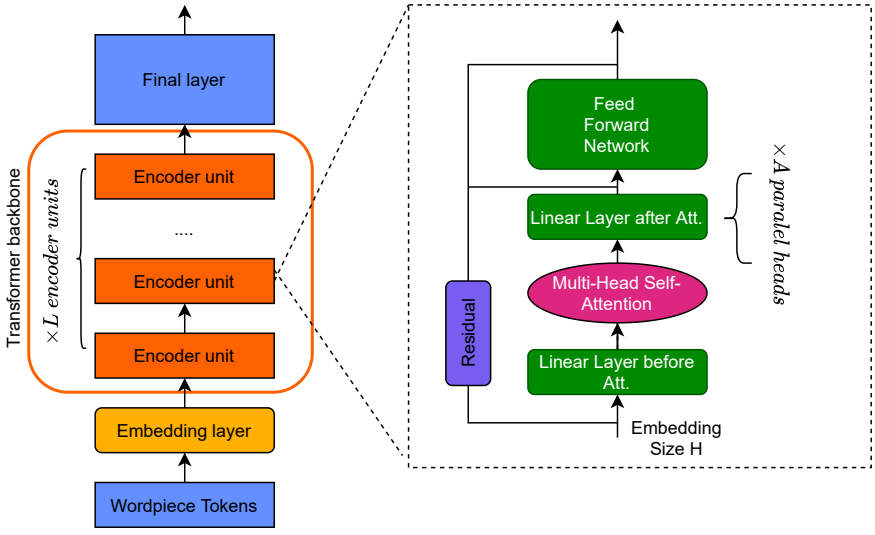


Figure 1.19: A figure inspired by (Ganesh et al., 2020) describes the BERT architecture in detail.

Original paper proposes two main architecture hyperparameters versions, *base* and *large* (see table 1.1), depending on the number of layers (L), the size of the hidden layer (H), and several heads in multi-head self-attention (A). Output before the classification head is a vector of size H for each of the input words.

Pre-training tasks

BERT is pretrained on two unsupervised tasks – Next Sentence Prediction (NSP) and Masked Language Model(MLM). These two tasks were selected because BERT's authors believe they force language models to learn general and valuable knowledge about language.

Next Sentence Prediction The input of the BERT model for this task is two sentences, A and B. In 50% of cases, sentence B is the sentence that follows sentence A in the source text. Otherwise, it is a random sentence from the corpus. A goal of the task is to decide whether the sentence B is the following one or a random one, i.e., binary classification. The motivation for this task is a need to represent relationships between sentences, not only between words. Experiments in (Devlin et al., 2019) showed its usefulness for text tasks as question answering. Sentence-level classification with BERT, as in the case of NSP, can be performed by using the last hidden representation of the CLS token (the first token of every input example) as an input into classification layer. Authors assumed that this token could work as a summary of the whole sentence, although later work has shown better approaches, i.e., (Liu et al., 2019).

Masked Language Modeling Masked Language Modeling, or in other words Cloze task (Taylor, 1953), consist of prediction of some missing words in the text. In BERT's case, its implementation follows: 15% of tokens in each sequence are chosen. For each of these chosen tokens, there is an 80% chance to be replaced by a MASK token, a 10% chance to be replaced by a random token, or it will remain unchanged with a 10% probability. This masking method ensures that the model will try to predict tokens not only in MASK token presence. For backpropagation, only predictions of the selected 15% tokens are taken into account. Prediction is made by a softmax function, whose input is the last hidden representation of the respective token, and the softmax layer outputs a probability distribution for predicting every possible word.

1.3.2 Derived models

After BERT, many other models built on similar architecture appeared. They all aim to improve the original BERT model in (at least) one of these three ways:

- A higher efficiency – original BERT models were trained for about four days on 4/16 TPU for base and large version respectively, and are quite memory intensive. Many methods for shortening the training time, memory consumption, or inference time, while preserving results, were successfully implemented - some of them even outperformed SOTA results set by BERT.
- An extension of applicability – BERT model works well for tasks requiring sentence or token classification, but is unable of language generation. The original model also does not offer a possibility of connecting knowledge out of the processed text. Both of these problems were explored with well-performing adjusted model architectures as a result (Zhang et al., 2019).

- Results improvements – Larger models, longer training times, more data, better pre-training objectives, and evolved architecture can lead to a significant improvement. Some models only demonstrate scalability with more data, while others win over BERT with more creative pre-training tasks or with a combination of many changes. There is another reason behind these models. It is not completely clear why BERT should work so well, so many authors offer a deep study of individual BERT components' impact on the performance, theoretical explanations, and possible improvements (Yang et al., 2019b; Liu et al., 2019).

The most famous and important BERT's derivates are presented in more detail in the following paragraphs.

XLNet (Yang et al., 2019b) solves two theoretical BERT's problems:

- MASK token, used in BERT's MLM objective, never occurs in real texts. Therefore, training data are substantially different from desired practical use; and
- BERT uses an assumption of independence between MASK tokens, given the unmask words, which does not hold. There definitely could be a strong relationship between two masked tokens in the sequence, even if they are not near each other. Moreover, it is desired for the model to learn such relationships.

XLNet presents three basic differences from BERT: presents a new training objective (permutation language modeling), uses Transforemers-XL (Dai et al., 2019) architecture instead of original Transformers encoders, and uses two-stream self-attention. The latter is a consequence of the used objective, which is briefly explained now. XLNet uses permutations of input sequence's tokens to model each token's probability, given the rest of the sequence (bidirectionally). To put it simple, a few of every possible permutation are selected, and then the model learns the probability of the word depending on the previous words in the permuted sentence. As any words are masked, we need to hide the content (segment and token embeddings) of the chosen token from the network, and keep position information. Two-stream self-attention processes these two kinds of information separately, so it is possible to mask out content information. XLNet presents new state-of-the-art results over previous BERT achievements – with the best models (trained on more data than BERT and four times more training time) and comparable settings (both model and training data sizes).

ERNIE (= Enhanced Language Representation with Informative Entities (Zhang et al., 2019)) enriches BERT with knowledge graphs. This contains knowledge presentation and a selection of objective, which is be able to work with knowledge as well as language information. As for encoding, ERNIE finds all named entities, links them into the knowledge graph, encodes knowledge graph using knowledge embeddings, which forms the input into ERNIE. BERT architecture with Transformer encoders is supplemented with knowledge encoders (K-encoders) stacked on the top of text encoders (T-encoders), and the output of the whole model are embeddings for words and entities. The pre-training objective

for language is the same as for BERT (NSP and MLM). The authors also present MLM-like objective for knowledge: masking of entities. This model outperforms BERT on knowledge-based tasks (entity typing and relation classification) and achieves results comparable to BERT on other NLP tasks.

RoBERTa (= Robustly optimized BERT approach (Liu et al., 2019)) falls into the third category of BERT family. Great improvements are the result of a thorough exploration of choices made for the original BERT model. RoBERTa uses larger models with more training data (145 GB of uncompressed text, much more than BERT, which uses only 16 GB) and longer training time, and offers at least a partial explanation of the influence of architecture, training settings, and objectives on BERT's success. The main differences from BERT are:

- removing NSP objective, which surprisingly increases the performance,
- FULL-SENTENCES sampling from the dataset: Every input sample contains full sentences sampled sequentially from data till the maximum size (512) is reached. Samples may cross boundaries of documents,
- bigger batch size ($2,048$ comparing to 256 of BERT)
- byte-level tokens encodings instead of WordPiece, with larger vocabulary size,
- dynamic masking: masks are not selected in advance before pre-training, but always created before data enters the model.

The resulting architecture is better than BERT, even trained on the same amount of data for the same amount of time.

UNiLM (= Unified Language Model (Dong et al., 2019)) enables BERT to generate text. Model architecture corresponds to $\text{BERT}_{\text{large}}$, but it combines bidirectional BERT training with objectives strengthening usage of a left context, which is important for language generation, and the model also utilizes NSP. The following rule selects the training objectives: 1/3 of the time, MLM objective (same as BERT) is used, 1/3 of the time the model uses sequence-to-sequence language modeling (using previous sentence and all words from left context), and unidirectional language modeling (right-to-left and left-to-right) objectives are used with 1/6 probability. Results are comparable to BERT, with no significant improvement, but the model is able to reach new state-of-the-art on several language generation tasks.

ELECTRA (= Efficiently Learning an Encoder that Classifies Token Replacements Accurately (Clark et al., 2020)). ELECTRA presents both more efficiency in training time and improvement on the pre-training task. The problem of BERT's MLM task as identified by ELECTRA is its waste of training data. As BERT masks 15% of tokens and only these tokens serve for model training, almost 85% of training data are unused. To solve this problem, ELECTRA proposes a new training task, which is defined over all input tokens – selected tokens are replaced by their alternatives (generated by an additional small network), and

the goal for each token is to decide whether it is an original or a replaced token. Therefore, the model has two parts – generator and discriminator (loosely inspired by Generative Adversial Networks). It significantly decreases training time, because data are used more efficiently. Even with shorter training time, ELECTRA outperforms both XLNet and RoBERTa.

T5 (= "Text-to-Text Transfer Transformer" (Raffel et al., 2019a)) is another approach to language generation and also another deep study of various parameters' influence. The training objective is similar to BERT; 15% of tokens are chosen and replaced with the mask token. One change is presented – the authors show that it is useful to select whole spans of the text of length three, and replace them with one mask token, rather than randomly selecting only words. Final architecture implements encoder-decoder pattern with five different model sizes. Only the two largest models achieved results comparable to BERT, and, in the case of the largest one, even outperformed previous state-of-the-art. T5 model uses text-to-text input format. Text-to-text format means that every input and also every output is in the text form. This is natural for some tasks like question answering or translation, but here also tasks with another output type are converted into text, and the format is unified for all tasks. The unified format allows the same architecture and training for all tasks and also easy multi-task training.

BART (= Bidirectional and Auto-Regressive Transformers) (Lewis et al., 2019) is an improvement over BERT, which offers generalization upon previous BERT-like models in terms of training objective and architecture. BART presents the good result in both classification and text generation tasks. BART, similarly to T5, implements encoder-decoder architecture, in this case uniting bidirectional encoder (like BERT) with left-to-right decoder (same as GPT). The main benefit of the BART architecture is that it allows arbitrary noising of the input text. Encoder first processes the noised input, and then decoder tries to reproduce the original (not damaged) text. Among many studied possible noises, sentence shuffling and text infilling proves to be the best, although the performance of various pre-training objectives varies for different tasks. The first objective changes the position of sentences. Text infilling replaces a randomly chosen span of text with the MASK token. The span's size is chosen from Poison distribution with λ equal to 3 (in contrast to T5, where the size was fixed) and can be zero. Size is comparable to BERT (10% more parameters). It reaches the performance of BERT and RoBERTa for comparable tasks,and presents a new state-of-the-art in language generation tasks.

Compression of BERT In addition to models mentioned above, there is a wide range of compression efforts. In Ganesh et al. (2020), authors offer an overview of possible compression methods:

- data quantization: using fewer bits to represent weights,
- various types of pruning: removing less important weights or components (encoder layers, attention heads),

- knowledge distillation: involving the large model as a teacher and a smaller model (possibly with completely different architecture) as a student. Student model can learn to imitate different settings of the teacher model (i.e., encoder outputs or output logits),
- architecture compression: sharing some weights across the model or decreasing the vocabulary size of embeddings.

This study shows a possibility of reducing BERT to the quarter of the original size, while preserving performance. One of the famous smaller models is ALBERT (=A Lite BERT (Lan et al., 2019)). ALBERT uses two reduction techniques – reduces the size of the embedding matrix by approximating it with two smaller matrices, and parameters sharing across the layers. In addition, ALBERT presents a modification to the NSP objective: sentence-order prediction (SOP). An input into this tasks consists of two sentences and the goal is to predict, whether they are in the correct ordering.

1.3.3 How to use language models

For using BERT-like models in new tasks, there are three possible ways (Liu et al., 2020):

- feature-based: one data pass through BERT to generate embeddings,
- fine-tuning: add new classification head(s) and train the whole model,
- adapter methods: adding task-specific layers between BERT layers and train them for target tasks with other BERT layers frozen (Stickland and Murray, 2019).

In the first case, the model is used only once to generate embeddings for all input data. These embeddings are stored and later fed to any other NLP model like regular embeddings. Technically, this can be achieved by stacking the second model on the top of BERT and freeze (disable training) the BERT layers, but it would be time and memory more consuming. This method is valid, and the results will be more likely improved by using BERT embeddings over non-contextualized embeddings. Even the BERT paper study shows that using a concatenation of the last four layers can achieve comparable performance to fine-tuned BERT, but this method still seems to lose some potential of language representations compared to learning the model on specific task (Sun et al., 2019).

Speaking about fine-tuning, there are many decision to be made, i.e., how to choose learning rate, whether to train all layers together or freeze some of them, if it is beneficial to use regularization, which layers to choose as an input into task-specific part of model, and many others. There is no guaranteed way, and it could also depend on the task type (sentence vs. token classification), but few typical possibilities, which usually work well, exist, and we now describe the main choices together with the commonly used approaches.

Sequence vs. Token Classification

Tasks which BERT can naturally solve fall into two categories – sequence classification and token classification. The majority of papers seem to focus on sequence classification of sentences or larger text parts, e.g., sentiment analysis, natural language inference, question answering, sentence similarity, etc. Token classification tasks classify each input token, i.e., word, word part, or punctuation mark. The type of task influences the resulting architecture and also brings various problems.

How to get knowledge from BERT?

Which information from BERT should go to classification layers is the fundamental question when designing a model. From the horizontal point of view, the CLS token is mostly used for a sequence classification, but utilizing some combination (e.g., concatenation, mean) of all sequence tokens is also possible (Rogers et al., 2020). Token classification uses an analogical approach – taking the first token of the word or combining all word's tokens seem both to work well with little to zero impact on the result (Kondratyuk and Straka, 2019; Kitaev et al., 2018). As for the vertical combinations, the original BERT paper proposes taking the last layer's representation of respective token(s). Furthermore, the combination of the last four layers is also used (the best option in Sun et al. (2019) was the max function). Generally spoken, usage of more layers proves to be advantageous, and it is possible to let the choice of layers to be learned (Yang and Zhao, 2019; Kondratyuk and Straka, 2019).

Learning dynamics

For the learning process itself, the first big question is whether to apply more pre-training, either on data for the task, domain or at least data for the same task as it showed to be beneficial (Sun et al., 2019). More technical details to be decided are the following:

Classification Heads It is possible to add only one simple classification head on the top of the BERT model, or maybe employ the more sophisticated network, e.g., some previous SOTA network, with BERT improving its inputs.

Layers Training Is it better to train the whole model, only some layers, or should it be dependent on the epoch number? Choosing the suitable scheme can improve the model and prevent catastrophic forgetting (Liu et al., 2020). One of the applicable approaches, proposed initially for different models than BERT, is gradual unfreezing (Howard and Ruder, 2018; Chronopoulou et al., 2019). As initially proposed, layers are unfrozen one by one during the training, e.g., one layer is added after each epoch. Because the last layers of the model contain the less general information (Howard and Ruder, 2018; Yosinski et al., 2014), it is appropriate to start with the last layer(s) and leave the other layers frozen. After some training, when the model is not a random mess and returns quite decent results, deeper model layers are further trained (fine-tuned) for few episodes and a significantly lower learning rate. This training practice's motivation is the belief that the original language model captured many generally applicable information

about the language, and is better than random initialization. Because the model is large and supervised data are usually not so big, it is undesirable to train the whole model, as it can take a long time to convergence, and randomly mess up with the trained model's knowledge. After the first training phase, however, the classification head knows the right direction, figuratively speaking, and with a small learning rate, the goal is to customize the language model a bit the for current task. In addition to gradual unfreezing, it is possible to divide the model only into two parts (BERT part and task-specific part) and train them gradually (Kondratyuk and Straka, 2019). Multi-stage layerwise training (Yang et al., 2020), designed for BERT, proposes training the output layer with only one encoder layer at a time, which leads to 25% faster training with good performance.

Learning rate Right learning rate choice can significantly improve the result. Absolute value of the learning rate for fine-tuning BERT usually lies between 3×10^{-6} and 5×10^{-5} (Devlin et al., 2019; Virtanen et al., 2019), (Kittask et al., 2020) as the higher learning rate can cause catastrophic forgetting (Sun et al., 2019). Together with its size, learning rate distribution over time and layers is important. Assigning lower learning rate to lower model layers corresponds to the same assumption as in the case of freezing the lower layers, i.e., it is desired to preserve the general information learned during pre-training, and it contributes to good results (Howard and Ruder, 2018; Sun et al., 2019), (Kondratyuk and Straka, 2019). Learning rate scheduling as slanted triangular learning rate (Howard and Ruder, 2018), inverse square root decay (used in (Kondratyuk and Straka, 2019; Raffel et al., 2019b) or linear decay (used in (Liu et al., 2019; Clark et al., 2020)) is an essential component of a successful training.

Last, is is also possible to select between different optimizers and different batch sizes. Optimizer seems not to be the most significant part of the decision, as most papers typically do not discuss this choice. Although, in (Chronopoulou et al., 2019), authors present the usage of two different optimizers for different layers – SGD for pre-trained model layers in order to preserve the learned knowledge, and Adam (Kingma and Ba, 2015) for the task-specific layer to support faster learning. BERT proved to be nicely scalable in selecting the batch size, and and usually the bigger the batch size the better the results (Liu et al., 2019).

1.3.4 Why BERT works?

It is quite common for deep learning architectures that the reason why they work (especially why they work so well compared to other possibilities) is not visible at first sight. The last part of this chapter focuses on some insight into the language models' functionality. When it comes to examining the inner working of BERT and similar models, two questions arise:

- What does the model know about the language?
- Where exactly in the model is all possessed knowledge stored?

What BERT knows about language?

There are several approaches to research the extent of information that BERT has. The experiments with shuffling or deleting some word show that BERT probably

does not rely too much on syntactic information (Ettinger, 2019; Rogers et al., 2020), although it is present there to some extent, as it is possible, for example, extract syntactic trees (Rosa and Mareček, 2019). BERT also contains semantic information of various types. Specifically, experiments indicate a presence of entity types, relations, or semantic roles (Tenney et al., 2019b). BERT possibly "knows" something about the real world, but is not able to perform complex reasoning above it (Rogers et al., 2020).

Where is knowledge stored?

While searching for the information encoding, it is possible to focus on attention heads or activations in layers. Tenney et al. (2019a) shows that different tasks are solved using different layers, specifically syntactic information is captured by lower layers (or middle layers; Rogers et al., 2020), and semantic information is spread over all layers. The inputs of the last BERT layer (usually used for downstream tasks) serve well as word embeddings, which corresponds with finding the clusters of these embeddings according to word meaning (Rogers et al., 2020). The last layers of BERT are also more task-specific, which leads to possible better transferability of middle layers. For attention heads, many papers show that original architecture with sixteen-headed multi-head attention is not optimal, and many heads can be removed in pre-trained models without losing the performance (Michel et al., 2019), which leads to a decrease of inference time. Even a reduction to only one head does not lead to a significant decrease in performance. However, a presence of more heads is better justified during pre-training, at least in early epochs, when the pruning of heads can decrease the performance significantly (Michel et al., 2019). Even though some heads can be redundant, other heads seem to capture specific syntactic relations, e.g, "objects of verbs, determiners of nouns, objects of prepositions, and coreferent mentions" according to Clark et al. (2019). The same authors also show that CLS token representation in the last layer attends to all words in the sequence, which justifies using these outputs for sequence classification.

Research of BERT's knowledge representation and storage, together with discoveries presented in the BERT-inspired models, agree that the original architecture leaves a lot of room for improvement or reduction, and that "BERT language skills are not as impressive as they seem" (Rogers et al., 2020).

2. Experiments

This chapter describes all experiments and their results. The first part is dedicated to presentation of different experiement hyperparameters, that are in many cases common to all tasks, followed by the description of each task and a discussion of results.

2.1 A description of training hyperparameters

2.1.1 General experiment setup (EXPE)

Training is performed in one of the following settings:

- **base**: Baseline implementation (described separately for each task, typically without using advanced language models).
- **ls**: This setup uses same setting as baseline implementation but with label smoothing.
- **embed**: BERT-like language model is used only to generate static embeddings in advance. Non-BERT part of the model is trained with BERT layers frozen (pre-trained, not changed during training).
- **full**: This options means training the whole model from the beginning (in contrast to *fine* option), but the classification head is not simplified (in contrast to *simple* option).
- **fine**: Fine-tuning consist of dividing the training time into two parts. First part of training is same to *embed* setting. In the second part, the whole model is trained together (as in *full*).
- **simple**: Model architecture is reduced to BERT layers with a simple classification head. This is a basic setting for all sentiment analysis experiments.[1]

2.1.2 Training data

Tagging and lemmatization tasks use the same set of data for all experiments, so there is no need for separate description. Sentiment analysis task, however, uses three possible options as a selection of training data:

- **mall—facebook—csfd**: Model is trained and evaluated on the (sub)set of Czech datasets.
- **zero**: Model is trained on English sentiment analysis dataset, but evaluated on Czech data.
- **eng**: Model is trained on the combination of Czech and English training data (and evaluated again on the Czech data).

[1] For tagging and lemmatization, all previously mentioned EXPE setups are performed with more sophisticated classification head than in *simple* version.

2.1.3 Learning rate scheduling type (LRTYPE)

Most experiments are expected to perform better with some kind of learning rate scheduling. This work implements three types of learning rate scheduling:

- **simple** *Simple* option indicates no more complex learning rate scheduling than setting different learning rates for different epochs in advance.
- **isrd** *isrd* means inverse square root learning rate decay defined by formula:

  $$1/\sqrt{n},$$

  where n is the current iteration.
- **cos**: Another learning rate scheduling used in this work is *cosine decay*, which applies the following formula:

  $$lr = lr^i_{min} + \frac{1}{2}\left(lr^i_{max} - lr^i_{min}\right)\left(1 + \cos\left(\frac{T_{curr}}{T_i}\pi\right)\right),$$

  where lr^i_{min} is the range of the learning rate, T_{curr} is the current epoch number, and T_i is the number of epochs after which the learning rate is restarted, i.e. increased to the lr^i_{max} value and T_{curr} is reset to 0.

Both *cos* and *isrd* are combined with *warmup*. Learning rate is linearly increasing for first k steps (one epoch in all experiments) from zero to the value in hyperparameters and than starts the decay.

2.1.4 Model layers selected for embeddings (LAYERS)

As discussed in the previous chapter, it is unclear how to extract best embeddings from the language model, especially which layers to take into account. According to the results published in (Devlin et al., 2019; Kondratyuk and Straka, 2019), we consider the following two promising approaches:

- **four**: Last four layers of the model are averaged to obtain final embeddings.
- **att**: Layer attention performs weighted sum of all model layers, and the weights are trained during training together with the rest of the model.

Experiments are also performed with different learning rates (LR), batch size (BATCH), and a number of epochs (EPOCH). Technical details needed for running scripts with the right arguments can be found in chapter 3.

2.1.5 Metrics

Metrics used for evaluation in this work are *accuracy* and *F1 score*. Accuracy is a percentage of correctly classified samples out of all samples, and it is the basic metric for all classification tasks (not only in this thesis). Accuracy can be sometimes misleading (Davis and Maiden), and there exist other metrics that can better reflect experimenter's goals. One of them is F1 score, which is used together

with accuracy for evaluation of sentiment analysis task due to comparability of results. F1 score is defined in terms of precision and recall. Precision

$$precision = \frac{TP}{TP + FP}$$

[2] express credibility of a positive result, e.g., if positive result means a need of surgery, it is definitely unwanted to have low precision and perform many dangerous and expensive surgeries unnecessarily. Recall, defined as:

$$recall = \frac{TP}{TP + FN},$$

, on the other hand tells us how many positives are captured. For example: *How likely I am to be pregnant with a negative pregnancy test?* F1 score formula for binary classification is than defined as

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}.$$

For multi-class classification, precision and recall needs to be redefined. F1 score can be computed per-class (for every class, binary classification of being in the class is taken). Per-class scores can be combined in one of following ways:

- macro-F1: average of per-class scores,
- weighted-F1: average as before, but weighted by the number of samples in each class,
- micro-F1: equals to accuracy.

[2]TP stands for true positive = number of samples correctly labelled as 1, FP (=false positives) are incorrectly labelled as 1, FN (=false negatives) is defined similarly.

2.2 Lemmatization and part-of-speech tagging

Lemmatization and POS tagging tasks are categorized as morphological analysis, share the same architecture and trained network and they will be described together in this section.

2.2.1 Task Definition

POS tagging
input: a sequence of words
output: tag (for each word), which contains not only part-of-speech (e.g. noun, pronoun, punctuation mark) but also other morphological analysis (case, tense, etc) corresponding to 15-places morphological tagging system by Hajič (2004). Description of each position can be found in table 2.1.

Lemmatization
input: a sequence of words
output: lemma (for each word) – a base form of a given words, for example nominative of singular for nouns or infinitive for verbs. In this work, lemmatization is treated as a classification problem with classes coresponding to generating rules which transform an input word into target lemma. For example of such rules see figure 2.1.

Metrics Accuracy is used for the evaluation and is reported separately for several options – only tags/lemmas, accuracy of joint classification of tags and lemmas, and also all three variants with an usage of a morphological dictionary (this option is described in more detail in 2.2.3).

Lemma Rule	Casing Script	Edit Script	Most Frequent Examples
↓0;d¦	all lowercase	do nothing	the→the to→to and→and
↑0¦↓1;d¦	first upper, then lower	do nothing	Bush→Bush Iraq→Iraq Enron→Enron
↓0;d¦-	all lowercase	remove last character	your→you an→a years→year
↓0;abe	all lowercase	ignore form, use be	is→be was→be 's→be
↑0;d¦	all uppercase	do nothing	I→I US→US NASA→NASA
↓0;d¦--	all lowercase	remove last 2 chars	been→be does→do called→call
↓0;d¦---	all lowercase	remove last 3 chars	going→go being→be looking→look
↓0;d--+b¦	all lowercase	change first 2 chars to b	are→be 're→be Are→be
↓0;d¦-+v+e	all lowercase	change last char to ve	has→have had→have Has→have
↓0;d¦---+e	all lowercase	change last 3 chars to e	having→have using→use making→make
↓0;d¦-+o→	all lowercase	change last but 1 char to o	n't→not knew→know grew→grow

Figure 2.1: Table 1 from (Straka et al., 2019c) presents 10 most common lemma generating rules in English EWT corpus. Each rule has two parts – a casing script for transforming uppercase and lowercase letters, and an edit script. The edit script can transform prefix, suffix, or also a root of the word. It uses the Wagner–Fischer algorithm (Wagner and Fischer, 1974), which finds the longest commont substring between the word and its lemma. Resulting rule is the shortest edit script converting the word into the lemma. More information can be found in (Straka et al., 2019c).

Position	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	Reserve
14	Reserve2	Reserve
15	Var	Variant, style

Table 2.1: Czech morphology developement is dated from 1989 (Hajič, 2004) and in description of words uses 15-places morphological tags as described in this table taken from `https://ufal.mff.cuni.cz/pdt2.0/doc/manuals/en/m-layer/html/ch02s02s01.html`. For more detailed description or for exploration of predictions given by this work is recommended to use website of Institute of Theoretical and Computational linguistics: `http://utkl.ff.cuni.cz/~skoumal/morfo/?pos=11&val=1`, although they use slightly different set with additional 16 position.

2.2.2 Related Work

Tagging

This work aims to improve previously published SOTA results for contextualized embeddings in Czech lemmatization and tagging (Straka et al., 2019a) and (Straka et al., 2021). POS tagging (for English) is dated back to 1971 with first rule-based approach on Brown Corpus (Greene and Rubin, 1971). Good results in POS tagging were achieved after year 2000 using both classical machine learning methods like Hidden Markov Models (Brants, 2000) or Support Vector Machines (Giménez and Màrquez, 2004), and perceptrons/neural networks (Collins, 2002). Actual English SOTA known to me is presented in Flair model (Akbik et al., 2018).[3] It is necessary to note that early papers had POS tagging defined differently than it is in this thesis. They focused only on selecting part of speech (noun, verb, etc...), meanwhile the later works (including this thesis) present complex morphological analysis.

One of the first automatic tagging experiments in Czech are described in (Hladká, 1998), which also shows differences between languages with rich inflexion (as Czech, but also Finnish or Turkish) and ones with simpler morphology (for example English or Spanish). Languages with complicated morphology have incomparably larger set of possible tags – English has less then one hundred of

[3]More detailed overvirew of English tagging can be found here: https://aclweb.org/aclwiki/

possible tags, Czech has almost 4,000 tags. Current SOTA results for tagging (and lemmatization) are presented in (Straka et al., 2021), which uses Czech version of RoBERTa model – RobeCzech. This is the model also used for some experiments in this work and, as expected, yields the best results. RobeCzech is based upon previous successful morphological analysis with contextual embeddings and BERT-like models (Straka et al., 2019c), (Straka et al., 2019b), (Straka et al., 2019a), (Straka, 2018) (all lastly mentioned models also achieved great results in lemmatization).

Although tagging is mostly considered to be a classification into predifined set of tags, the sets themselves can vary. Penn treebank uses a tagset of 54 different tags, which presents parts of speech and additional information like tense or number.[4] There are some differences between this tagset and other English datasets or taggers (e.g., TreeTagger (Schmid, 1995) or CLAWS tagset (Chapelle, 1988)). All English tagsets are really small compared to languages like Czech or Turkish. As mentioned before, Czech uses 15-positioning tags, which is a natural solution for such type of languages. These positions can be predicted together or for each position separately. The first approach creates big tagset but guarantees consistency among positions (e.g. there will be no tense for a noun or a case for a verb). In the case of separate prediction, each position can be treated as a classification problem separately, which causes problems, because the individual parts of tag are not independent. Better approach is to use sequence-to-sequence modelling (Sutskever et al., 2014), which outputs the tag as a sequence of positions and takes into account previously generated position as in (Malaviya et al., 2019).

Lemmatization

Lemmatization (both Czech and English) has undergone a similar development as tagging, starting with rule-based approaches and statistical approaches (Plisson et al., 2004), continuing with neural networks and recently achieving good results with BERT-like models (Kondratyuk and Straka, 2019). Lemmatization is typically performed as a sequence-to-sequence model, therefore it takes a word as a sequence of characters and produces a new sequence of characters, which is the lemma. This approach is teoretically better than classification into rules, because it is possible to generate every existing lemma. However, it can generate simply every possible character sequence, which may not be an existing word. Lemmatization as a classification task into edit scripts set firstly appeared in Chrupala et al. (2008) and was explored further by Straka (2018). Sequence-to-sequence model can be also used for production of edit rules (same rules as used in this work)(Chakrabarty et al., 2017), (Müller et al., 2015) and (Yildiz and Tantuğ, 2019).

2.2.3 Dataset and Preprocessing

Dataset for these tasks is taken from data of Prague Dependency Treebank (PDT) (PDT35), version 3.5 from year 2018. Data consists of sentences with lemmas and tags. For ambiguous words, data contain all possible analyses, which were

[4] see: `https://www.sketchengine.eu/penn-treebank-tagset/`

generated using morphoDita (Straková et al., 2014) and morphological dictionary (Hajič and Hlaváčová, 2016) (described later).[5] For example, Czech word "psa" has one possible lemma ("pes") but two possible tags, because it could be one of two possible grammatical cases – genitive or accusative. Input data for such word looks as follows:

```
psa pes NNMS2-----A---- NNMS4-----A----.
```

Data contains about 1,600 unique tags and about 1,500 different lemma rules. The number of lemmas is significantly smaller than a number of unigue lemmas (72, 000) (Straková et al., 2014) or tags, because words with similar morphological function have same way of creating lemma from the word, e.g. words *malého* (=little, accusative, sg, m.,) and *červeného* (=red, accusative, sg, m.,) have the same lemma rule:

```
↓ 0;d¦---+ý+-+1.
```

Dataset is originally divided into tree parts - train, development and test, which is also used in this work. Input sentences are preprocessed as follows:

- mapping characters and words into numbers – Mapping words/characters, which were found in train dataset into integers (from one to the number of unique words). This means that the network has no information about words/characters which appears only in test or development dataset. All newly appeared words/characters are mapped into one same number (typically 0) for *UNK* token/character.
- tokenization – Tokenizer for corresponding BERT-like model transforms input words into tokens. Each word is transformed into one or more strings, which are converted into numbers. This serves as an input into BERT part of the model. To create these input embeddings, the whole sentence for each word is needed as the same words can have different representation in different contexts. More information can be found in section 1.3.1.

2.2.4 Architecture and Experiments

The model for lemmatization and tagging is build upon a model (and a code) for previous work on Czech NLP processing with contextual embedding (Straka et al., 2019a). Data preprocessing is taken over from the paper as well as the structure of the lemmatizer and the tagger network, which is extended by BERT-like models, hoping for improvements. Previous work showed that training tagging and lemmatization together in one network can be mutually advantageous, so both of these analyses are an output of one network, and are trained jointly. Detailed visualisation of network architecture can be found in figure 2.2.

The architecture of the network can be divided into three parts – inputs, optional RNNs, classification head:

[5] genarator of analyses is available online: `https://lindat.mff.cuni.cz/services/morphodita/`.

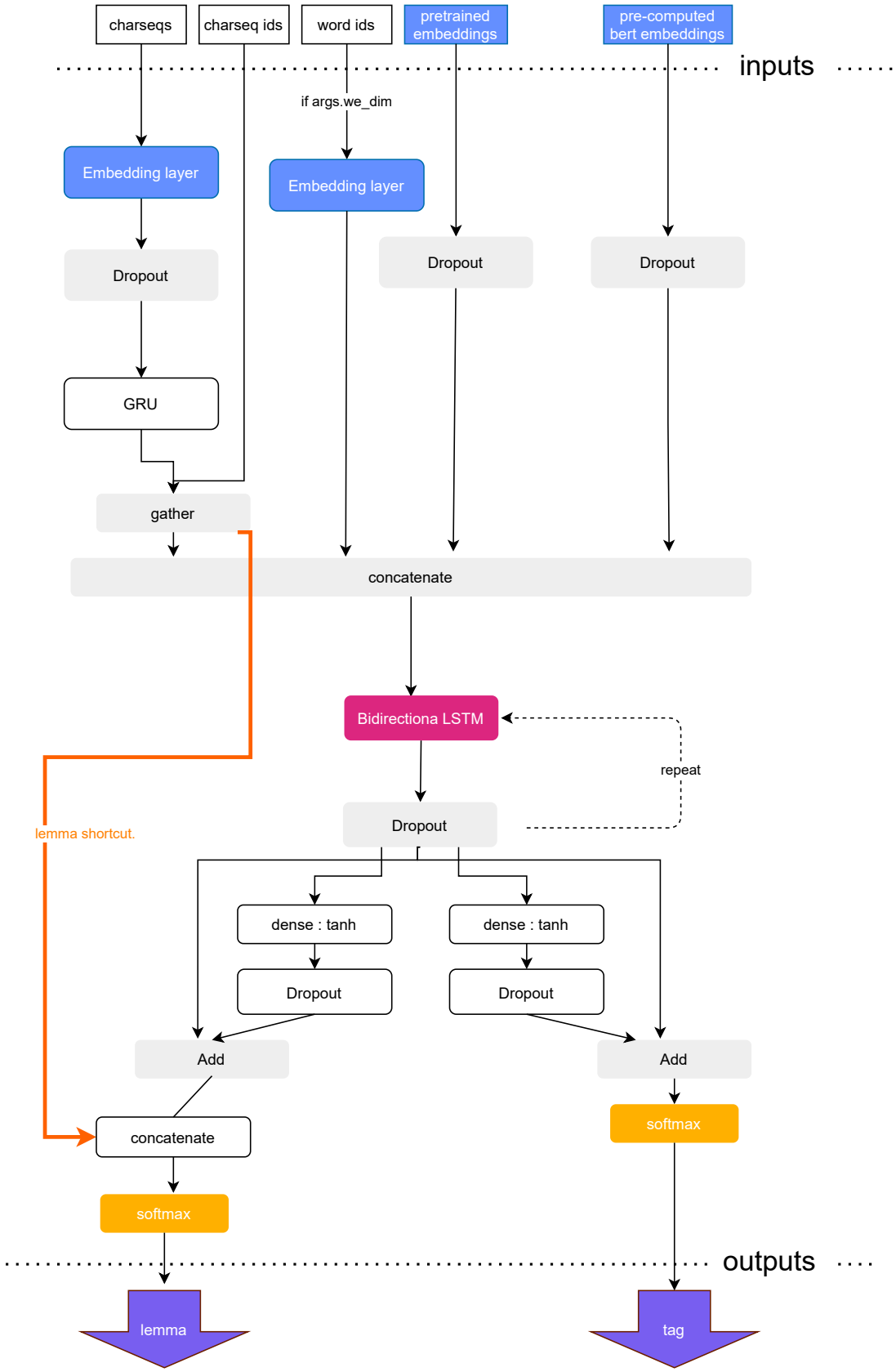


Figure 2.2: Tagging and lemmatization joint model architecture.

Inputs An input set of the network consists of five different input types – characters (charseqs), words (charseq ids), correct responses (word ids), pretrained embeddings, and possibly precomputed BERT embeddings (depending on the experiment type). Two other types of embeddings are created before the further processing of inputs by RNN cells: character-level embeddings and another word embeddings that are, in contrast to BERT and pretrained embeddings, also trained during the training process.

RNN cells Character-level embeddings are further processed via 1 layer of Gated Recurrent Unit (GRU) and all inputs (or their embeddings) are processed by recurrent part of network (specifically by three layers of Long Short-Term Memory (LSTM) cells).

Classification head(s) After the processing by recurrent neural networks, network employs two separate classification heads, one for tagging and another for lemmatization. Both heads use dense layer with tanh activation function to allow task-specific non-linear transformation as used in (Straka, 2018) and a softmax function for obtaining the probability distribution over target classes. Lemmatization, however, presents another change – addition of character level data without RNN processing, that are used together with the rest of the values as an input into the softmax following (Straka, 2018), as it leads to better performance of lemmatization in the case of shared network between both tasks.

Morphological Dictionary All classification can be done with or without use of a morphological dictionary MorfFlex (Hajič and Hlaváčová, 2016), that can provide possible pairs *tag-lemma*. If used, the generated tag and lemma is a pair with maximal likelihood, but chosen just from the dictionary. This leads to more consistent results.

Experiments

This part uses all main **experiment types** as decribed in 2.1: *base, ls, embed, fine, simple*, and *full*. Three **BERT-like models** are used for experiment setup:

- multilingual BERT (mBERT) (Devlin et al., 2019),
- XLM-RoBERTa (Conneau et al., 2019),
- RobeCzech (Straka et al., 2021).

XLM-RoBERTa and mBERT are trained on 100/104 different languages including Czech. RobeCzech is a recently published version of RoBERTa, trained only on Czech data. XLM-RobERTa is used only for embedding and one version of fine-tuning, and this model was omitted in other experiements because of weak results and high computational complexity. There exists another monolingual Czech model, Czert (Sido et al., 2021), which uses the original BERT architecture and was outperformed by RobeCzech (Straka et al., 2021).

A selection of layers is made in both ways – last four layers (*four)* and learning of weighted sum of all layers (*att*). The layer attention is made only for the fine-tuning setup, and as the weighted sum does not show a significant benefit, mean of the last four layer is the only method used for other experiments.

Learning rate is used as usual for each type of task and three different learning rate schedules were applied in each combination of hyperparameters: cosine decay *(cos)*, inverted square root decay *(isrd)* and a one epoch warm-up followed by a constant learning rate *(warmup)* inspired by (Kondratyuk and Straka, 2019) and (Howard and Ruder, 2018). For *embed* experiments, *warmup* is replaced by a simple division of training into two parts with different learning rates.

hyperparametr	value
beta_2	0.99
optimizer	Adam
cle_dim	256
dropout	0.5
label smoothing	0.3
rnn_cell	LSTM
rnn_cell_dim	512
rnn_layers	3
we_dim	512
word_dropout	0.2
batch size	64

Table 2.2: Hyperparameters of tagging and lemmatization common to all experiments (if they make sense in the context of experiments).

Batches have size 64, given by the compromise between the pursuit of relatively big batch size and computational resources. Summary of hyperparameters, which do not differ across experiments is presented in table 2.2. Other hyperparameters for each experiment are in table A.1.

Reimplementation of (Straka et al., 2019b) without any BERT-like model incorporation serves as a baseline.

2.2.5 Results and Discussion

Best presented model (experiment no. 18) achieved the same or better results than the current state-of-the-art tagging and lemmatization results (table 2.3). Complete results are in table 2.4. Experiment *tl_18* is the version with fine-tuning, Czech monolingual model, RobeCzech, and without layer attention, although the difference from comparable experiment with layer attention is insignificant and can be just accidental. The dominance of the Czech model was expected and additional expert knowledge contained in the complicated architecture was also assumed to be better. Experiments also showed that fine-tuning approach achieves better results than full training from the beginning. This may be due to the choice of hyperparameters, especially the learning rate, but the standard ones were selected, implying that at best, it is more difficult to find the right parameters for*full* and *simple* variants. Although *simple* experiments presents standard approach of using pretrained BERT models, they turned out being less

successful even than *embed* experiments, that are faster to train and less memory intensive.

Experiment	Without Dictionary			With Dictionary		
	Tags	Lemmas	Both	Tags	Lemmas	Both
(Straka et al., 2019a)	*97.94*	*98.75*	*97.31*	*98.05*	*98.98*	*97.65*
StrakaC	*97.67*	*98.63*	*97.02*	*97.91*	*98.94*	*97.51*
RobeCzech	*98.43*	*98.79*	*97.83*	*98.50*	***99.00***	*98.11*
baseline	97.04	98.56	96.41	97.31	98.83	96.90
emb(12)	98.38	98.79	97.80	98.48	98.99	98.10
best(18)	**98.50**	**98.80**	**97.90**	**98.57**	**99.00**	**98.19**

Table 2.3: *Straka2019C* is a comparable solution (BERT embeddings only) from (Straka et al., 2019a) to *emb*, which was transformed into Tensorflow 2 in this work as a *baseline*. *Emb* is a solution with static BERT embeddings and *best(18) is the best resulting model in this thesis (experiment id = 18).*

Error Analysis

This section offers a little exploration of differences in error across models. This comparison includes three models:

- tl_18 – the best model in tagging and lemmatization,
- tl_3 – the best model with mBERT,
- tl_1 – the baseline model with label smoothing.

best vs. baseline The best model (tl_18) improves prediction in 3,247 tags and is worse in 421 tag predictions. More than 80% of newly correctly predicted tags are composed by three parts of speech: NN (noun), AA (adjective), and RR (preposition). Table A.2 presents improved tags with a frequency at least 10. The most frequent tag (`NNIS1-----A----`) presents proper names of places (e.g. Jersey, Tenesee), but we can see that other frequent tags are nominatives and accusatives of masculinum, singular, inainamate (cs: *rod mužský neživotný*) or femininum, plural. These two cases have the same form for mentioned categories, so they are indistinguishable without context, and that is where BERT showed to be very useful. The same situation is with adjectives, again mostly nominative or accusative of the same form, for example words *další* (following) or *stínový* (shadowy). The third category are prepositions that can be connected with both accusative and dative as *na* (on), or *o* (about).

mBERT vs. RobeCzech Best mBERT is better in 468 tags and worse in 1,703 tags than the best model. The most frequent tags improved by *tl_18* are similar to previous comparison. Nominative and accusative are again the most common cases improved, but the differences between these two models are not so significant. This results in verbs appearing higher in the table of most frequent tags, although the absolute value of better predictions on verbs is similar to

Model		EXPE	EP	LAYERS	LR	Lemmas		Tags		Both	
						Raw	Dict	Raw	Dict	Raw	Dict
0	NA	base	A	NA	simple	98.58	98.81	97.05	97.31	96.43	96.9
1	NA	ls	B	NA	simple	98.55	98.81	97.12	97.34	96.51	96.94
2	mBERT	embed	B	four	simple	98.69	98.93	97.83	97.98	97.17	97.58
3			C	four	cos	98.74	98.95	97.91	98.04	97.28	97.63
4			C	four	isrd	98.73	98.94	97.89	98.02	97.28	97.61
5	xlm-Roberta		B	four	simple	98.57	98.8	97.33	97.54	96.68	97.12
6			C	four	cos	98.6	98.83	97.45	97.62	96.81	97.21
7			C	four	isrd	98.59	98.83	97.44	97.61	96.81	97.2
8	RoBECzech		B	four	simple	98.77	98.97	98.38	98.48	97.78	98.08
9			C	four	cos	98.79	98.99	98.38	98.48	97.80	98.10
10			C	four	isrd	98.78	98.98	98.4	98.48	97.8	98.09
11	mBERT	fine	D	four	simple	98.69	98.93	97.84	97.99	97.21	97.59
12			E	four	cos	98.72	98.95	97.97	98.08	97.33	97.68
13			E	four	isrd	98.68	98.9	97.72	97.86	97.09	97.46
14	xlm-Roberta		D	four	simple	98.62	98.84	97.72	97.9	97.07	97.48
15			E	four	cos	98.67	98.9	97.95	98.09	97.32	97.69
16			E	four	isrd	98.63	98.85	97.66	97.83	97.03	97.41
17	RoBECzech		D	four	simple	98.78	98.98	98.46	98.55	97.86	98.16
18			E	four	cos	**98.80**	**99.00**	**98.50**	**98.57**	**97.90**	**98.19**
19			E	four	isrd	98.76	98.95	98.33	98.41	97.72	98.02
20	mBERT	fine att	D	att	simple	98.67	98.91	97.76	97.92	97.13	97.52
21			E	att	cos	98.72	98.95	97.98	98.1	97.34	97.69
22			E	att	isrd	98.67	98.91	97.69	97.85	97.05	97.45
23	xlm-Roberta		D	att	simple	98.6	98.81	97.62	97.77	96.96	97.35
24			E	att	cos	98.67	98.89	97.91	98.06	97.29	97.66
25			E	att	isrd	98.65	98.86	97.65	97.81	97.03	97.41
26	RoBECzech		D	att	simple	98.77	98.97	98.38	98.47	97.79	98.08
27			E	att	cos	**98.8**	98.99	98.47	98.54	97.88	98.16
28			E	att	isrd	98.77	98.96	98.33	98.41	97.72	98.01
29	mBERT	simple	F	four	warmup	98.17		97.32		96.46	
30			G	four	cos	98.15		97.39		96.47	
31			G	four	isrd	98.13		97.12		96.29	
35	RoBECzech		F	four	warmup	98.49		98.28		97.41	
36			G	four	cos	98.46		98.30		97.39	
37			G	four	isrd	98.59		98.27		97.53	
38	mBERT	full	F	four	warmup	98.16	98.86	97.35	97.79	96.46	97.34
39			G	four	cos	98.04	98.85	97.36	97.81	96.3	97.34
40			G	four	isrd	98.22	98.86	97.34	97.73	96.46	97.29
44	RoBECzech		G	four	warmup	98.49	98.95	98.21	98.34	97.38	97.93
45			G	four	cos	98.25	98.95	98.17	98.33	97.08	97.89
46			G	four	isrd	98.55	98.99	98.19	98.35	97.39	97.95

Table 2.4: This table presents complete results for tagging and lemmatizationt tasks. Column EP presents number of epochs and corresponding learning rates are explained in Attachement A.1

previous comparison. In both situations, improved verbs predictions relate mostly to verbs with the same form in singular and plural of the third-person, e.g. *vyváží* (exports) or *stojí* (stands). The complete table of the most frequent tags is available in table A.3.

2.3 Sentiment Analysis

As stated in Veselovská (2017): "Sentiment analysis, also known as opinion mining, is an automatic detection of a positive or negative polarity, or neutrality of ... a text sequence", which is exactly as the sentiment analysis is understood in this work. There are, however, some other definitions consisting of e.g. opinion extraction, irony, or stance (Montoyo et al., 2012) and sentiment analysis can also continue with e.g., opinion extraction. Another tasks related to sentiment analysis is subjectivity analysis (whether the presented opinion is objective or highly subjective), which is also not included in this work, mainly because of the lack of labelled data for Czech. It is possible to analyze individual expressions, sentences, or whole documents (Veselovská, 2017). This thesis focuses on the document-level classification, which has many real-life use cases and Czech training data are available.

2.3.1 Task definition

Sentiment analysis
input: sequence of sentences (a whole post or comment, depending on the source)
output: prevailing sentiment of the input from categories: neutral, positive, negative.

Metric For evaluating performance, two metrics are used: weighted-F1 score and accuracy. Accuracy is a standard metric for classification and weighted-F1 allows better comparability and also provides additional insights into models evaluation.

2.3.2 Related Work

As every languge-related task, sentiment analysis is best explored for English. It is possible to derive sentiment by supervised learning (typical are Support Vector Machines or Maximum Entropy classifier) or using rule-based approach – vocabulary of emotionally coloured words, emoticons etc (Çano and Bojar, 2019; Veselovská, 2017). BERT-like models were succesfully used to improve result for sentiment analysis task on English (Devlin et al., 2019) and also other languages, for example Estonian (Kittask et al., 2020), Indonesian (Putra et al., 2020) ,or Italian (Pota et al., 2021).

There are not so many attempts to sentiment analysis in Czech in comparison to English, however some attempts were made with both neural networks and traditional machine learning - Naive Bayes Classifiers, Support Vector Machines, and Maximum-Entropy-based classifiers (Veselovská, 2017). A thorough study of supervised machine learning methods on *mall* and *facebook* dataset is offered in Çano and Bojar (2019). For a practical use, Žižka and Dařena (2010) present automatic sentiment prediction of unlabelled text based on a small set of labelled patterns via searching similarities. As the neural networks dominate in many NLP taks, they are also applied in sentiment analysis. One of the first attepts to apply neural networks on Czech sentiment is described in Lenc and Hercig (2016), which evaluates besides others all three datasets used in this work on document-level

sentiment analysis. Kyselý (2017) perform sentiment analysis using embeddings and convolutional neural network on multidimensional embedding, which is quite unusual as CNNs are typically used for image processing. Kyselý (2017) use same three datasets, but classify only on sentence-level (they filter out longer samples), which is simpler as longer texts tend to be more inconsistent about sentiment (Veselovská, 2017). Libovický et al. (2018) present state-of-the art results in three Czech NLP tasks including sentiment analysis. They use only CSFD dataset with resulting accuracy 80.8%, which is comparable to previous SOTA (Brychcín and Habernal, 2013). The second mentioned paper uses quite complicated method for classification incorporating the fact of which movie is reviewed, while Libovický et al. (2018) use only bidirectional LSTMs with multiple attention heads following state-of-the-art results on English (Lin et al., 2017). There are five previous works know to me, which involves BERT-like models in Czech sentiment:

- XLM-Roberta applied on all three datastets trimmed to 128 characters[6]
- Klouda et al. (2019) apply multilingual BERT on the mall dataset with resulting accuracy about 81%, which did not outperform the naive Bayes classifier baseline with 84% accuracy,
- Sido et al. (2021) present monolingual Czech model Czert, based on BERT and ALBERT models, and evalutes it on csfd and facebook datasets with new state-of-the-art results,
- Straka et al. (2021) publish another monolingual model, based on more succesfull RoBERTa model, and surpassed Czert on the facebook dataset.

2.3.3 Dataset and Preprocessing

Four main Czech datasets with sentiment annotation are available: news from Aktualne.cz (aktualne) (Veselovská, 2017), user reviews from MALL.cz (mall), film reviews from csfd.cz (csfd), and posts from Czech branch pages on facebook.cz (facebook) (the last tree datasers are introduced in Habernal et al., 2013). As *aktualne* dataset turned out to be problematical because the text were ambiguous even for annotators, and its authors later used other mentioned datasets (Veselovská, 2017), this work also focuses only on the three other data sources – *mall*, *csfd* and *facebook*.[7] Some experiment are also performed with in-domain training on English data. The *imdb* dataset[8] is used for this purpose. This dataset contains movie reviews from the biggest movie rating website imdb.com. This leads to some problems described later in this section, because English dataset contains only binary classification (positive/negative). Table 2.5 summarizes each dataset. We randomly split all dataset into train, development, and test datasets with the same labels distribution as the original datasets, similarly to Sido et al. (2021).

As can be seen in figure 2.3, distribution of labels differs among datasets. Moreover, figure 2.4 shows that the resulting dataset is highly unbalanced, which

[6] `http://www.janpalasek.com/sentiment-analysis-czech.html`

[7] All three datasets are all available here: `http://liks.fav.zcu.cz/sentiment/`

[8] `https://www.tensorflow.org/datasets/catalog/imdb_reviews`

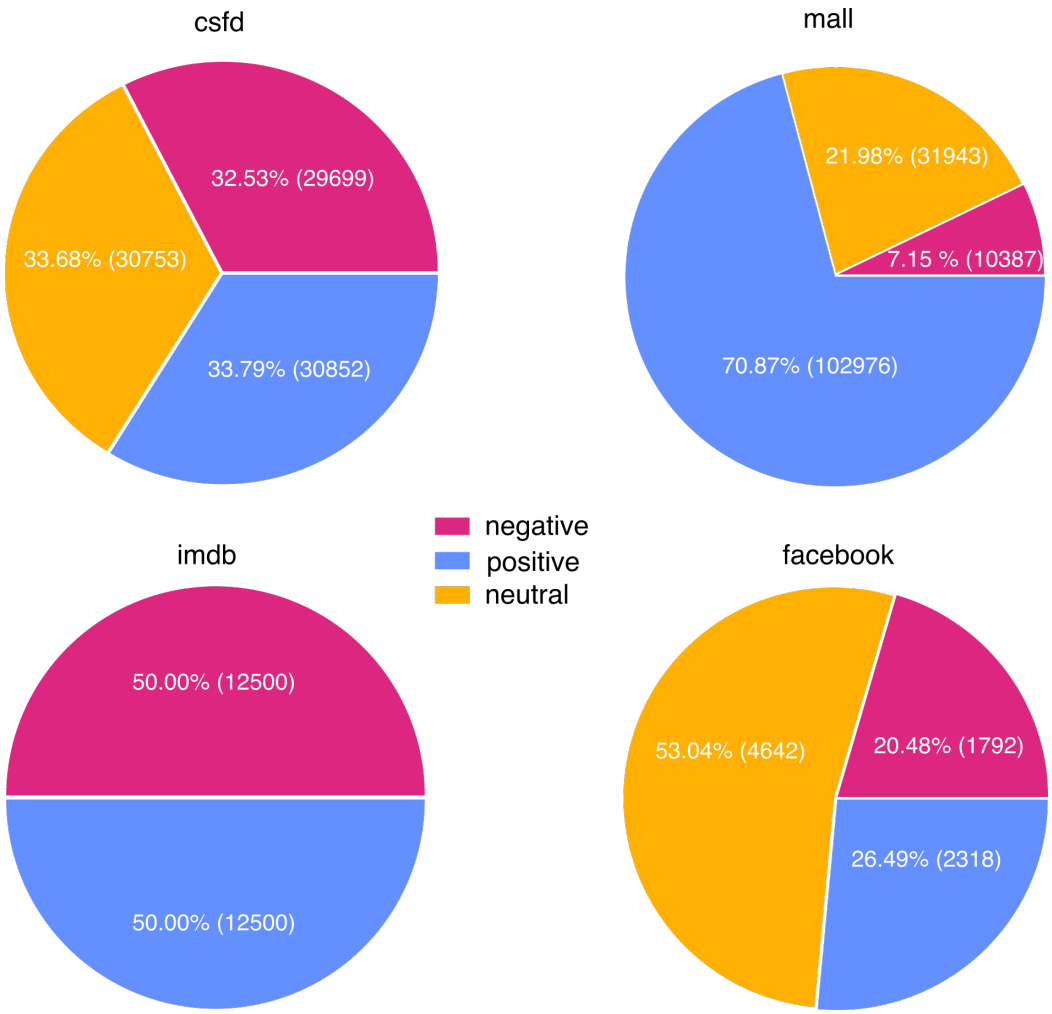


Figure 2.3: Distribution of positive/neutral/negative labels in each dataset.

may causes divergence during training. Due to the big part of labels being positive, many learning strategies end with predicting only *positive* class, i.e., 55% accuracy, so unfortunately learning nothing.

2.3.4 Experiments and Architecture

The main division of experiments is by the input dataset – each of Czech models separately and one joint dataset consisting of all Czech datasets, i.e., four different datasets. All variants perform both layers attention and an average of last four layers. As for the learning rate, all experiments were made with learning rate $3 \cdot 10^{-5}$, and there are always two types of learning rate decay – cosine and inverse square root decay.

The network architecture is much simpler than in the tagging and lemmatization task, and corresponds to *simple* setting of these tasks – only BERT-like model and a classification head consisting of a dense layer with softmax activation function.

Baseline for these models is a Naive Bayes Classifer (NB) with term frequency–inverse document frequency (tf–idf) representation. *tf–idf* for a word is defined

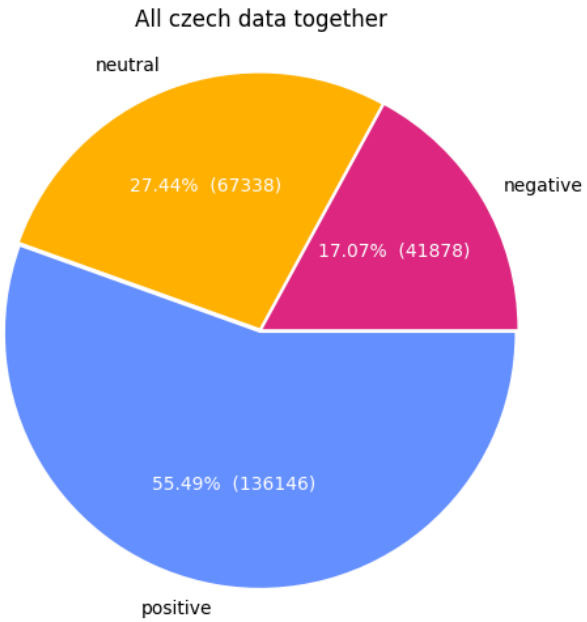


Figure 2.4: Percentage and absolute values of labels in all three Czech datasets together.

	length	labels	domain
mall	145306	positive neutral negative	domestic appliance reviews
csfd	91304	positive neutral negative	movie reviews
facebook	9752	positive neutral negative	brand pages of e.g. shops or mobile network providers
imdb	25000	positive negative	movie reviews

Table 2.5: Three Czech datasets (mall, facebook, csfd) and one English (imdb) are used for training in this work.

in this way: *tf* stands for a term frequency

$$tf = \frac{word\ occurences}{number\ of\ words\ in\ document}$$

and this is count over the whole dataset, while *idf* is inverse document frequency

$$idf = \frac{number\ of\ documents}{number\ of\ documents\ with\ word}.$$

The *Idf* acts as an evaluation of the importance of the word. The result is then:

$$tf - idf = tf \cdot idf.$$

This representation serves as an input into Naive Bayes Classifier. Naive Bayes Classifier (NB; Duda et al., 1973), a probabilistic model, which models probability of the class k given the data features x_i: $p(C_k|x_1, ..., x_n)$ and uses *naive* assumption of features to be conditionally independent given the class C_k.

2.3.5 Results and Discussion

Models based upon RoBECzech not only outperformed the baseline, but also achieved new state-of-the-art results in all three datasets as can be seen in table 2.6. Complete results can be seen in table 2.9. The best model for each dataset was the one trained on that dataset, although model trained on joint datasets performed comparable to single-data model on *csfd* and *mall*. For *facebook*, joint model was worse by 9%. This can be caused by the a difference in distribution of labels between datasets. *Facebook* dataset has the most different label distribution in comparison to *all_czech* together (mostly neutral posts vs. mostly positive, see figure 2.3 and figure 2.4).

Following Kyselý (2017), resulting models are evaluated on five different Czech sentences to manifest the differences between models (table 2.7). It can be seen that predicting neutral vs. negative is still tricky for models, which can be also seen in confusion matrices (table 2.8). Confusion matrices shows that predicting neutral is complicated in general, meanwhile models have learned to distinguish well between positive and negative sentiment. Table 2.7 also shows that *csfd* model is quite different from the rest. It is probably caused by the difference in the training data nature.

Because the BERT model was trained on multilingual data, it is naturally not so good in a language sparsely presented in its training data. When transferring the learned knowledge to Czech sentiment task, we actually want to improve model in two ways: teach it something more specific about the given task, i.e., sentiment, and improve its knowledge about the used language (Czech in this case). By using Czech sentiment dataset, both aspects are incorporated into training. To obtained better results and following Putra et al. (2020), we also included English sentiment dataset *imdb* during training. The idea behind is that BERT is quite good in English and maybe can learn useful knowledge about the given task from data in a more familiar language. In the table of results there are two additional experiment types – *zero* and *eng*. Zero stands for zero-shot and the model is trained only on English data, but evaluated on the joint Czech dataset. This, of course, does not returns results competitive to models trained on Czech data, however table 2.10 shows that zero models actually learned some useful knowledge about the task, which they could apply to Czech data. *Zero(2)* improved by 40% after training on English data only. *Zero(35)* is the model with RobeCzech, therefore it has the better prior knowledge of Czech, but it can also be improved by training on English data within the task. In Putra et al. (2020), authors use this approach because of the lack of the data in Indonesian, so they divided training into two parts. Firstly, they train models with different approach including zero-shot, and then they chose the best model for fine-tuning on Indonesian data. As we have the Czech monolingual BERT-like model, we did not continue by fine-tuning the *zero* experiments, because they did not seem promising.

dataset	models	Acc	F1-w	F1-m
All Czech	baseline	82.00	70.00	-
	(Kyselý, 2017)	*67.82*	*67.00*	-
	best(16)	**84.04**	**83.86**	**80.84**
csfd	baseline	69.07	69.00	-
	Czert	-	-	*84.79*
	(Habernal et al., 2013)	-	-	79.00
	(Kyselý, 2017)*	*71.34*	*71.00*	-
	best(16)	84.02	84.00	-
	best(69)	**84.89**	**84.87**	**84.83**
mall	baseline	84.72	83.00	-
	(Kyselý, 2017)	*82.52*	*81.00*	-
	(Habernal et al., 2013)	-	-	75.00
	(Klouda et al., 2019)(Bert)	81.00	79.00	-
	(Klouda et al., 2019)(SVM)	84.00	82.00	-
	best(16)	84.40	84.00	-
	best(63)	**84.60**	**84.14**	**76.85**
facebook	baseline	67.30	63.00	-
	RobeCzech	-	-	*80.13*
	(Habernal et al., 2013)	-	-	69.00
	XLM-RoBERTa**	-	-	***82.29***
	Czert	-	-	*76.55*
	(Kyselý, 2017)	*71.62*	*71.00*	-
	best(16)	75.00	74.98	-
	best(45)	**81.80**	**81.65**	80.11

Table 2.6: Best results for all datasets and a comparison to previous work. Best(16) is a best model for joint dataset and best(x) is always the best model for respective dataset. Numbers in italics are from related work. Related work results except *Czert* are 10-fold crossvalidation results. $\star$ (Kyselý, 2017) performs only sentence-level classification. $\star\star$ This is the large XLM-RoBERTa model from Straka et al. (2021), which is four times larger than the BERT base model.

Input	joint	mall	facebook	csfd
Rozbila se po prvním použití, je na hovno. *It broke after the first use, it is shitty.*	Neg	Neg	Neg	Neut
Rozbila se až za rok. *It broke after a year of use.*	Neg	Neg	Neg	Neut
S manželem jsme si víkend moc užili. *Me and my husband enjoyed the weekend.*	Pos	Pos	Pos	Pos
Ok, ale nic zajímavého. *Ok, but nothing interesting.*	Neut	Neg	Neg	Neg
super zboží *super product.*	Pos	Pos	Pos	Neut

Table 2.7: Evaluation of models on four Czech sentences. *mall* model was not included as a separate option, as the best model is the joint one.

Combined datasets (16)

True labels \ Predicted labels	neut	neg	pos
neut	7027	885	2025
neg	1122	4783	307
pos	1279	207	18857

mall (63)

True labels \ Predicted labels	neut	neg	pos
neut	2896	199	1696
neg	344	1082	132
pos	932	54	14461

csfd (69)

True labels \ Predicted labels	neut	neg	pos
neut	3600	685	170
neg	603	3769	241
pos	146	225	4257

facebook (45)

True labels \ Predicted labels	neut	neg	pos
neut	440	41	51
neg	57	135	7
pos	23	3	243

Table 2.8: Confusion matrices for best model in each category.

Model	Acc	F1
zero(2) before training	21.33	14.68
zero(2) after training	49.51	44.67
zero(35) before training	34.78	31.80
zero(35) after training	58.13	50.89

Table 2.10: Results of selected zero-shot experiments.

MODEL		EXPE	LAYERS	LRTYPE	Acc	F1-w	F1-m
1	mBERT	czech	four	isrd	80.89	80.62	76.89
2		zero			49.51	44.67	35.91
3		eng			81.17	80.90	77.45
4		czech		cos	82.56	82.35	79.10
5		zero			53.41	47.64	38.49
6		eng			82.55	82.37	79.12
13	RoBECzech	czech	four	isrd	81.17	80.90	79.65
14		zero			55.31	48.26	38.79
16		czech		cos	84.04	83.86	80.72
17		zero			57.64	48.79	39.03
19	mBERT	czech	att	isrd	81.61	81.43	78.02
20		zero			53.92	47.55	38.23
21		eng			81.79	81.32	77.78
22		czech		cos	82.62	82.42	79.11
23		zero			51.99	46.63	37.59
24		eng			82.59	82.36	79.08
31	RoBECzech	czech	att	isrd	83.26	83.18	80.06
32		zero			58.36	50.40	40.89
34		czech		cos	83.88	83.68	80.57
35		zero			58.13	50.89	35.71
37	mBERT	facebook	four	isrd	75.30	74.97	35.71
38				cos	76.20	75.89	73.32
41	RoBECzech			isrd	80.10	79.87	77.98
43				cos	81.50	81.37	79.90
44					81.00	80.78	79.02
45					81.80	81.65	80.11
46	mBERT		att	isrd	76.40	75.67	72.68
47				cos	77.20	76.83	74.20
50	RoBECzech			isrd	79.60	79.07	76.78
51				cos	80.60	80.38	78.76
52	mBERT	mall	four	isrd	82.80	82.80	74.94
53				cos	84.27	83.88	76.48
56	RoBECzech			isrd	83.17	83.37	76.00
57				cos	84.73	84.30	76.95
58	mBERT		att	isrd	83.02	82.90	75.36
59				cos	84.04	83.61	75.94
62	RoBECzech			isrd	84.08	83.88	76.18
63				cos	84.60	84.14	76.85
64	mBERT	csfd	four	isrd	80.77	80.83	80.79
65				cos	82.04	82.04	82.01
68	RoBECzech			isrd	83.06	83.05	83.00
69				cos	84.89	84.87	84.83
70	mBERT		att	isrd	81.63	81.60	81.57
71				cos	82.20	82.19	82.16
74	RoBECzech			isrd	83.13	83.18	83.13
75				cos	84.32	84.32	84.28

Table 2.9: This table presents complete results on the sentiment task. Presented metrics are accuracy, macro-F1, and weighted-F1.

3. Implementation analysis

The main purpose of this chapter is to offer the technical description of the code accompanying this work for better reproducibility and possible further experiments on every of the presented tasks. This chapter describes an implementation of all language models, other related code, and also presents all used libraries and technologies.

All code forms an attachment of this work and is also publicly available on GitHub.[1] Experiment were performed on the Artificial Intelligence Cluster (AIC)[2] provided by the Institute of Formal and Applied Linguistics, Charles University.[3]

3.1 Code description

This section describes the code – technologies and hardware used for experiments, where to find the scripts for replicating the experiments, and how to run them.

3.1.1 Technologies description

All code is implemented in Python (v3.6.9). Python is a popular language for machine learning, because of easy use and many available libraries, which allows to focus on high-level problem solving instead of technical details. All dependencies and used libraries are listed in the `/code/requirements.txt` file, but we also mention the most important libraries explicitly.

TensorFlow and Keras

The main library used for developing deep learning models in this work is Tensorflow (Abadi et al., 2015). This library provides lots of tools for machine learning, especially for neural networks. Keras is a wrapper library over Tensorflow and provides easy use of the most common machine learning scenarios (Chollet, 2015). Tensorflow together with PyTorch (Paszke et al., 2019) is probably the most frequently used library for deep learning, both providing similar functionality. The reason behind this choice of Tensorflow is the fact that this thesis builds on the previous work and uses code developed in Tensorflow.

Transformers

As mentioned in other part of text, this work reuse pretrained language models based on BERT. Transformers library from Hugging Face(Wolf et al., 2019) contains many variants of pretrained BERT models and tools for their usage as tokenizers or learning rate schedulers.

[1] `https://github.com/flower-go/DiplomaThesis`
[2] `https://aic.ufal.mff.cuni.cz/`
[3] `https://ufal.mff.cuni.cz/home-page`

Pandas

Pandas library (Reback et al., 2020) serves well for data analysis as it provides data structures like DataFrame, which provides named columns, advanced data indexation, selection, merging, joining, reshaping and other functionality similar to tools provided by e.g., SQL databases. It does not only provide a rich set of tools, but they are also developed with an emphasis on performance.

Scikit-learn

Scikit-learn (Pedregosa et al., 2011) is another useful Python library specialized on machine learning. In contrast to TensorFlow, scikit-learn focuses on classical machine learning, not on neural networks, providing all important variants of machine learning models as well as supporting tools for training, e.g., cross validation or various metrics.

Numpy

Numpy (Harris et al., 2020) is a library providing powerful multidimensional arrays with many predefined operations. It is fast and it is a de-facto standard library for numerical operations over number arrays.

Jupyter Notebook

Jupyter notebook (Kluyver et al., 2016) is a web application for development. In this work, Jupyter notebook is used for providing the trained models for exploration. Jupyter suits well for this purpose because it, in addition to a possibility of running a separate parts of code in different cells, also supports visualisations and Markdown formatted text and it can be useful especially for explanatory purposes.

3.1.2 Code Structure

Each task has code in a separate directory. All code for tagging and lemmatization is placed in the folder `morphodita_research`. Main files are `morpho_tagger_2.py` and `bert_fine-tuning_simple.py`, which serves for running all experiments relating to tagging and lemmatization. Script arguments are described in more detail in tables 3.1, 3.2 and 3.3.

Sentiment analysis experiments (in folder `sentiment`) can be run using `sentiment_analysis.py` with arguments as described in tables 3.1 and 3.4.

3.1.3 Working example

The best model is publicly available in the Git repository[4] and licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Licence.[5] If you want to try the model prediction or see a working example of usage of such models, you can use a public Google Colaboratory (Bisong, 2019) Jupyter

[4] `https://github.com/flower-go/DiplomaThesis/tree/master/code/morphodita-research/models`

[5] `https://creativecommons.org/licenses/by-nc-sa/4.0/`

Argument	Values	Description
accu	int	Accumulation of gradient. Effective batch size is batch_size times accu.
batch_size	int	Batch size (without accumulation).
bert	string	Name of the bert model (from the HuggingFace library) or path to the model.
checkp	String	Name of the saved model weights. Saving weights is used instead of saving he whole model.
debug	0/1	Debug mode loads small debug data if available.
label_smoothing	decimal number	Coefficient for label smoothing.
dropout	float	Dropout amount applied on various places of the network.
epochs	"x:l1,y:l2"	This will perform x epochs with learning rate l1 and y epochs with learning rate l2.
layers	None/"att"	If "att", all BERT-like model layers are combined with learned weights.
warmup_decay	None /"i:x"/"c:x"/"n:x"	If not None, training will incorporate inverse square root decay, cosine decay, or warm-up for x episodes.
fine_lr	float	Different learning rate for the classification head.

Table 3.1: A list of arguments common to all scripts.

notebook, which downloads all necessary data and returns predictions for a given text. This notebook si available here: `https://colab.research.google.com/github/flower-go/DiplomaThesis/blob/master/PlayWithModels.ipynb`.

Argument	Values	Description
beta_2	float	An argument for the optimizer.
cle_dim	int	Dimension of character-level embeddings.
exp	string	Name of logs files.
factors	"Lemmas,Tags"	Factors to be predicted – Lemmas, Tags, or both.
word_dropout	float	Probability of masking a word in the sentence during training.

Table 3.2: A list of arguments common to both scripts for tagging and lemmatization.

Argument	Values	Description
data	string	Input data directory. Data are supposed to be divided into train, dev and test `.txt` files.
char_dropout	float	Dropout for characters.
embeddings	string	Path to pre-comuputed embeddings to use.
factor_layers	int	Number of dense-and-dropout blocks for each of factors.
lemma_re_strip	string	Regular expression for suffix to be stripped from lemma.
lemma_rule_min	int	Minimal occurences to keep a lemma rule.
predict	string	Produce only a prediction with the model from the path given in this argument.
rnn_cell	"LSTM"/"GRU"	Type of RNN cell to use.
rnn_cell_dim	int	Dimension for RNN cells.
rnn_layers	int	Number of recurrent cell layers.
we_dim	int	Dimension of trainable word embeddings.
bert_model	string	Trained checkpoint for loading. Training will continue from this checkpoint.
test_only	string	Path to the model, which will be loaded and weights will be printed.

Table 3.3: A list of arguments specific to `morpho_tagger_2.py`, with detailed description.

Argument	Values	Description
datasets	{"mall,csfd,facebook"}	Names of the input Czech datasets, separated by comma.
english	float	A percentage training data which should be taken from the English IMDB dataset.
freeze	0,1	Value 1 means that BERT layers will not be trained.
seed	int	Inicialization of random seed.
kfold	"k:i"	Data will be splitted into k folds and the i-th fold will be used for evaluation. It serves for running k-fold cross-validation in parallel runs.

Table 3.4: Arguments for `sentiment_analysis.py` script.

Conclusion

In this thesis, we implemented Czech part-of-speech tagging, lemmatization and sentiment analysis with the usage of Bidirectional Encoder Representations from Transformers-like architectures. We achieved state-of-the-art results in tagging (accuracy 98.57%) and lemmatization (accuracy 99.00%) and joint accuracy of these two tasks 98.19%, which presents the error reduction of 67% for tagging accuracy and 53% for lemmatization accuracy compared to previous publicly available model (Straková et al., 2014).[6] We also presents new state-of-the-art results in sentiment analysis on two used datasets – *mall* and *csfd*. We also explored various training techniques and showed the good performance of static embeddings compared to any further training of BERT models. This thesis also examines types of errors BERT helps to solve. All code, text and best models are publicly available on GitHub: `https://github.com/flower-go/DiplomaThesis`.

Future Work

The BERT-like models ale able to transfer knowledge even across very different languages, and also previous work suggest, that training on languages from similar family can improve results in all included languages (Arkhipov et al., 2019), therefore one possible improvement can be in both pre-training the new BERT-like model (similarly to (Straka et al., 2021)) on joint dataset for i.e., Czech, Slovak and Polish, or use such multilingual data for task-specific training, for example in sentiment analysis. In addition, results with large XLM-Roberta suggests that selecting this architecture for pretraining could be advantageous despite computational demands. Sentiment analysis data are quite small, although they could be retrieved from pairs (review – star rating) and many Czech companies have access to such data. Experiments with tagging and lemmatization also show good results of precomputed embedding approach. The question therefore arises as to whether it would not be better to simply add precomputed BERT embeddings into existing sentiment analysis solutions.

[6] The best publicly available model is to our best knowledge available at `https://ufal.mff.cuni.cz/morphodita/users-manual#czech-morfflex-pdt_model`, which has 95.55% tagging and 97.96% lemmatization accuracy.

Bibliography

[PDT35] : *PDT 3.5 Main page.* – URL `https://ufal.mff.cuni.cz/pdt3.5`

[Abadi et al. 2015] Abadi, Martín ; Agarwal, Ashish ; Barham, Paul ; Brevdo, Eugene ; Chen, Zhifeng ; Citro, Craig ; Corrado, Greg S. ; Davis, Andy ; Dean, Jeffrey ; Devin, Matthieu ; Ghemawat, Sanjay ; Goodfellow, Ian ; Harp, Andrew ; Irving, Geoffrey ; Isard, Michael ; Jia, Yangqing ; Jozefowicz, Rafal ; Kaiser, Lukasz ; Kudlur, Manjunath ; Levenberg, Josh ; Mané, Dan ; Monga, Rajat ; Moore, Sherry ; Murray, Derek ; Olah, Chris ; Schuster, Mike ; Shlens, Jonathon ; Steiner, Benoit ; Sutskever, Ilya ; Talwar, Kunal ; Tucker, Paul ; Vanhoucke, Vincent ; Vasudevan, Vijay ; Viégas, Fernanda ; Vinyals, Oriol ; Warden, Pete ; Wattenberg, Martin ; Wicke, Martin ; Yu, Yuan ; Zheng, Xiaoqiang: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. – URL `http://tensorflow.org/`. – Software available from tensorflow.org

[Akbik et al. 2018] Akbik, Alan ; Blythe, Duncan ; Vollgraf, Roland: Contextual String Embeddings for Sequence Labeling. In: *Proc. 27th Int. Conf. Comput. Linguist.* (2018)

[Allen 1987] Allen, Robert B.: Several Studies on Natural Language · and Back-Propagation. 1987. – Forschungsbericht

[Arkhipov et al. 2019] Arkhipov, Mikhail ; Trofimova, Maria ; Kuratov, Yuri ; Sorokin, Alexey: Tuning Multilingual Transformers for Language-Specific Named Entity Recognition, Association for Computational Linguistics (ACL), sep 2019, S. 89–93. – URL `https://github.com/google-research/`

[Bahdanau et al. 2014] Bahdanau, Dzmitry ; Cho, Kyunghyun ; Bengio, Yoshua: NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE. 2014. – Forschungsbericht

[Bengio et al. 2003] Bengio, Yoshua ; Ducharme, Réjean ; Vincent, Pascal ; Jauvin, Christian ; Ca, Jauvinc@iro U. ; Kandola, Jaz ; Hofmann, Thomas ; Poggio, Tomaso ; Shawe-Taylor, John: A Neural Probabilistic Language Model. 2003. – Forschungsbericht. – 1137–1155 S

[Bird et al. 2009] Bird, Steven ; Klein, Ewan ; Loper, Edward: *Natural Language Processing with Python*. 1st. O'Reilly Media, Inc., 2009. – ISBN 0596516495

[Bisong 2019] Bisong, Ekaba: *Google Colaboratory*. S. 59–64, 09 2019. – ISBN 978-1-4842-4469-2

[Brants 2000] Brants, Thorsten: TnT - A Statistical Part-of-Speech Tagger. In: *CoRR* cs.CL/0003055 (2000). – URL `https://arxiv.org/abs/cs/0003055`

[Brown et al. 1990] Brown, Peter F. ; Cocke, John ; Della Pietra, Stephen A. ; Della Pietra, Vincent J. ; Jelinek, Fredrick ; Lafferty, John D. ; Mercer, Robert L. ; Roossin, Paul S.: A STATISTICAL APPROACH TO MACHINE TRANSLATION. In: *Comput. linquistics* 16(2) (1990), S. 79–85

[Brown et al. 2020] Brown, Tom B. ; Mann, Benjamin ; Ryder, Nick ; Subbiah, Melanie ; Kaplan, Jared ; Dhariwal, Prafulla ; Neelakantan, Arvind ; Shyam, Pranav ; Sastry, Girish ; Askell, Amanda ; Agarwal, Sandhini ; Herbert-Voss, Ariel ; Krueger, Gretchen ; Henighan, Tom ; Child, Rewon ; Ramesh, Aditya ; Ziegler, Daniel M. ; Wu, Jeffrey ; Winter, Clemens ; Hesse, Christopher ; Chen, Mark ; Sigler, Eric ; Litwin, Mateusz ; Gray, Scott ; Chess, Benjamin ; Clark, Jack ; Berner, Christopher ; McCandlish, Sam ; Radford, Alec ; Sutskever, Ilya ; Amodei, Dario: *Language models are few-shot learners.* 2020

[Brychcín and Habernal 2013] Brychcín, Tomáš ; Habernal, Ivan: *Unsupervised Improving of Sentiment Analysis Using Global Target Context.* 2013. – URL `https://aclanthology.org/R13-1016`

[Çano and Bojar 2019] Çano, Erion ; Bojar, Ondřej: Sentiment Analysis of Czech Texts: An Algorithmic Survey. In: *ICAART 2019 - Proc. 11th Int. Conf. Agents Artif. Intell.* 2 (2019), jan, S. 973–979. – URL `http://arxiv.org/abs/1901.02780http://dx.doi.org/10.5220/0007695709730979`

[Chakrabarty et al. 2017] Chakrabarty, Abhisek ; Pandit, Onkar A. ; Garain, Utpal: Context sensitive lemmatization using two successive bidirectional gated recurrent networks. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, S. 1481–1491

[Chapelle 1988] Chapelle, C.: The Computational Analysis of English—A Corpus-Based Approach. In: *TESOL Quarterly* 22 (1988), S. 668–669

[Cheng et al. 2016] Cheng, Jianpeng ; Dong, Li ; Lapata, Mirella: Long Short-Term Memory-Networks for Machine Reading. URL `https://arxiv.org/abs/1601.06733`, 2016. – Forschungsbericht

[Cho et al. 2014] Cho, Kyunghyun ; Van Merriënboer, Bart ; Gulcehre, Caglar ; Bahdanau, Dzmitry ; Bougares, Fethi ; Schwenk, Holger ; Bengio, Yoshua: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: *EMNLP 2014 - 2014 Conf. Empir. Methods Nat. Lang. Process. Proc. Conf.*, Association for Computational Linguistics (ACL), jun 2014, S. 1724–1734. – URL `https://arxiv.org/abs/1406.1078v3`. – ISBN 9781937284961

[Chollet 2015] Chollet, François: *keras.* `https://github.com/fchollet/keras`. 2015

[Chronopoulou et al. 2019] Chronopoulou, Alexandra ; Baziotis, Christos ; Potamianos, Alexandros: An Embarrassingly Simple Approach for Transfer

Learning from Pretrained Language Models. In: *NAACL HLT 2019 - 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf.* 1 (2019), feb, S. 2089–2095. – URL http://arxiv.org/abs/1902.10547

[Chrupala et al. 2008] Chrupala, Grzegorz ; Dinu, Georgiana ; Genabith, Josef van: Learning Morphology with Morfette. In: *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*. Marrakech, Morocco : European Language Resources Association (ELRA), Mai 2008. – URL http://www.lrec-conf.org/proceedings/lrec2008/pdf/594_paper.pdf

[Clark et al. 2019] Clark, Kevin ; Khandelwal, Urvashi ; Levy, Omer ; Manning, Christopher D.: What Does BERT Look At? An Analysis of BERT's Attention. In: *arXiv* (2019), jun. – URL http://arxiv.org/abs/1906.04341

[Clark et al. 2020] Clark, Kevin ; Luong, Minh-Thang ; Le, Quoc V. ; Manning, Christopher D.: ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In: *arXiv Prepr.* (2020), mar. – URL http://arxiv.org/abs/2003.10555

[Collins 2002] Collins, Michael: Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In: *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, Association for Computational Linguistics, Juli 2002, S. 1–8. – URL https://aclanthology.org/W02-1001

[Conneau et al. 2019] Conneau, Alexis ; Khandelwal, Kartikay ; Goyal, Naman ; Chaudhary, Vishrav ; Wenzek, Guillaume ; Guzmán, Francisco ; Grave, Edouard ; Ott, Myle ; Zettlemoyer, Luke ; Stoyanov, Veselin: Unsupervised Cross-lingual Representation Learning at Scale. In: *arXiv* (2019), nov. – URL http://arxiv.org/abs/1911.02116

[Dai and Le 2015] Dai, Andrew M. ; Le, Quoc V.: Semi-supervised Sequence Learning. URL http://ai.stanford.edu/amaas/data/sentiment/index.html, 2015. – Forschungsbericht

[Dai et al. 2019] Dai, Zihang ; Yang, Zhilin ; Yang, Yiming ; Carbonell, Jaime ; Le, Quoc V. ; Salakhutdinov, Ruslan: Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. In: *ACL 2019 - 57th Annu. Meet. Assoc. Comput. Linguist. Proc. Conf.* (2019), jan, S. 2978–2988. – URL http://arxiv.org/abs/1901.02860

[Davis and Maiden] Davis, Kirk ; Maiden, Rodney: Original Paper The Importance of Understanding False Discoveries and the Accuracy Paradox When Evaluating Quantitative Studies.

[Devlin et al. 2019] Devlin, Jacob ; Chang, Ming W. ; Lee, Kenton ; Toutanova, Kristina: BERT: Pre-training of deep bidirectional transformers for language understanding. In: *NAACL HLT 2019 - 2019 Conf. North Am.*

Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf. Bd. 1, Association for Computational Linguistics (ACL), 2019, S. 4171–4186. – ISBN 9781950737130

[Dong et al. 2019] Dong, Li ; Yang, Nan ; Wang, Wenhui ; Wei, Furu ; Liu, Xiaodong ; Wang, Yu ; Gao, Jianfeng ; Zhou, Ming ; Hon, Hsiao-Wuen: Unified Language Model Pre-training for Natural Language Understanding and Generation. In: *arXiv* (2019), may. – URL http://arxiv.org/abs/1905.03197

[Duda et al. 1973] Duda, Richard O. ; Hart, Peter E. ; Stork, David G.: *Pattern classification and scene analysis*. Bd. 3. Wiley New York, 1973

[Ettinger 2019] Ettinger, Allyson: What BERT is not: Lessons from a new suite of psycholinguistic diagnostics for language models. In: *arXiv* (2019), jul. – URL http://arxiv.org/abs/1907.13528

[Feijo and Moreira 2020] Feijo, Diego de V. ; Moreira, Viviane P.: Mono vs Multilingual Transformer-based Models: a Comparison across Several Language Tasks. In: *arXiv* (2020), jul. – URL http://arxiv.org/abs/2007.09757

[Forcada and Ñeco 1997] Forcada, Mikel L. ; Ñeco, Ramón P.: Recursive hetero-Associative memories for translation. In: *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)* Bd. 1240 LNCS, Springer Verlag, 1997, S. 453–462. – URL https://link.springer.com/chapter/10.1007/BFb0032504. – ISBN 3540630473

[Francis and Kucera 1979] Francis, W. N. ; Kucera, H.: Brown Corpus Manual / Department of Linguistics, Brown University, Providence, Rhode Island, US. URL http://icame.uib.no/brown/bcm.html, 1979. – Forschungsbericht

[Ganesh et al. 2020] Ganesh, Prakhar ; Chen, Yao ; Lou, Xin ; Khan, Mohammad A. ; Yang, Yin ; Chen, Deming ; Winslett, Marianne ; Sajjad, Hassan ; Nakov, Preslav: Compressing Large-Scale Transformer-Based Models: A Case Study on BERT. In: *arXiv* (2020), feb. – URL http://arxiv.org/abs/2002.11985

[Giménez and Màrquez 2004] Giménez, Jesús ; Màrquez, Lluís: SVMTool: A general POS tagger generator based on Support Vector Machines. In: *Proceedings of the 4th LREC*. Lisbon, Portugal, 2004

[Goldberg 2015] Goldberg, Yoav: A Primer on Neural Network Models for Natural Language Processing. URL http://www.cs.biu., 2015. – Forschungsbericht

[Goodfellow et al. 2016] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016

[Google] Google: *Google AI Blog: A Neural Network for Machine Translation, at Production Scale*. – URL https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html. – Zugriffsdatum: 2020-10-25

[Greene and Rubin 1971] Greene, B.B. ; Rubin, G.M.: *Automatic Grammatical Tagging of English*. Department of Linguistics, Brown University, 1971. – URL https://books.google.cz/books?id=VznTygAACAAJ

[Habernal et al. 2013] Habernal, Ivan ; Ptáček, Tomáš ; Steinberger, Josef: Sentiment Analysis in Czech Social Media Using Supervised Machine Learning. In: *Proceedings of the 4th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*. Atlanta, Georgia : Association for Computational Linguistics, June 2013, S. 65–74. – URL http://www.aclweb.org/anthology/W13-1609

[Hajič 2004] Hajič, J.: *Disambiguation of Rich Inflection: Computational Morphology of Czech*. Karolinum, 2004. – URL https://books.google.cz/books?id=sB63AAAACAAJ. – ISBN 9788024602820

[Hajič and Hlaváčová 2016] Hajič, Jan ; Hlaváčová, Jaroslava: *MorfFlex CZ 161115*. 2016. – URL http://hdl.handle.net/11234/1-1834. – LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University

[Harris et al. 2020] Harris, Charles R. ; Millman, K. J. ; Walt, St'efan J. van der ; Gommers, Ralf ; Virtanen, Pauli ; Cournapeau, David ; Wieser, Eric ; Taylor, Julian ; Berg, Sebastian ; Smith, Nathaniel J. ; Kern, Robert ; Picus, Matti ; Hoyer, Stephan ; Kerkwijk, Marten H. van ; Brett, Matthew ; Haldane, Allan ; R'io, Jaime F. del ; Wiebe, Mark ; Peterson, Pearu ; G'erard-Marchant, Pierre ; Sheppard, Kevin ; Reddy, Tyler ; Weckesser, Warren ; Abbasi, Hameer ; Gohlke, Christoph ; Oliphant, Travis E.: Array programming with NumPy. In: *Nature* 585 (2020), September, Nr. 7825, S. 357–362. – URL https://doi.org/10.1038/s41586-020-2649-2

[Hewitt and Liang 2020] Hewitt, John ; Liang, Percy: Designing and interpreting probes with control tasks. In: *EMNLP-IJCNLP 2019 - 2019 Conf. Empir. Methods Nat. Lang. Process. 9th Int. Jt. Conf. Nat. Lang. Process. Proc. Conf.*, Association for Computational Linguistics, 2020, S. 2733–2743. – ISBN 9781950737901

[Hladká 1998] Hladká: Part of Speech Tags for Automatic Tagging and Syntactic Structures 1. 1998. – Forschungsbericht

[Hochreiter and Urgen Schmidhuber 1997] Hochreiter, Sepp ; Urgen Schmidhuber, J J.: Long short-term memory. URL http://www7.informatik.tu-muenchen.de/{~}hochreithttp://www.idsia.ch/{~}juergen, 1997 (8). – Forschungsbericht. – 1735–1780 S

[Hoerl and Kennard 1970] Hoerl, Arthur E. ; Kennard, Robert W.: Ridge Regression: Biased Estimation for Nonorthogonal Problems. In: *Technometrics* 12 (1970), Nr. 1, S. 55–67. – ISSN 15372723

[Howard and Ruder 2018] Howard, Jeremy ; Ruder, Sebastian: Universal language model fine-tuning for text classification. In: *ACL 2018 - 56th Annu. Meet. Assoc. Comput. Linguist. Proc. Conf. (Long Pap.* Bd. 1, 2018

[Huh et al. 2016] Huh, Minyoung ; Agrawal, Pulkit ; Efros, Alexei A.: What makes ImageNet good for transfer learning? 2016. – Forschungsbericht

[Hutchins] Hutchins, John: *Two precursors of machine translation: Artsrouni and Trojanskij*. – URL https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.2564{&}rep=rep1{&}type=pdf. – Zugriffsdatum: 2020-10-23

[Hutchins 1996] Hutchins, John: ALPAC: the (in)famous report. The MIT Press, 1996. – Forschungsbericht. – 131–135 S. – URL https://books.google.com/books?hl=cs{&}lr={&}id=yx3lEVJMBmMC{&}oi=fnd{&}pg=PA131{&}dq=alpac+report{&}ots=se2vhONMHp{&}sig=ByL2IgJLxRwF3f6n9bqOPFx88r4

[Jurafsky and Manning 2012] Jurafsky, Dan ; Manning, Christopher: Natural language processing. In: *Instructor* 212 (2012), Nr. 998, S. 3482

[Kingma and Ba 2015] Kingma, Diederik P. ; Ba, Jimmy L.: Adam: A method for stochastic optimization. In: *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, International Conference on Learning Representations, ICLR, dec 2015. – URL https://arxiv.org/abs/1412.6980v9

[Kitaev et al. 2018] Kitaev, Nikita ; Cao, Steven ; Klein, Dan: Multilingual Constituency Parsing with Self-Attention and Pre-Training. In: *ACL 2019 - 57th Annu. Meet. Assoc. Comput. Linguist. Proc. Conf.* (2018), dec, S. 3499–3505. – URL http://arxiv.org/abs/1812.11760

[Kittask et al. 2020] Kittask, Claudia ; Milintsevich, Kirill ; Sirts, Kairit: Evaluating Multilingual BERT for Estonian. In: *arXiv* (2020), oct. – URL http://arxiv.org/abs/2010.00454

[Klouda et al. 2019] Klouda, Ing K. ; Langr, Lukáš ; Daniel Vašata, Ing: *Title: Product review sentiment analysis in the Czech language Student*, Czech Technical University in Prague, Bachelor's thesis, 2019

[Kluyver et al. 2016] Kluyver, Thomas ; Ragan-Kelley, Benjamin ; Pérez, Fernando ; Granger, Brian ; Bussonnier, Matthias ; Frederic, Jonathan ; Kelley, Kyle ; Hamrick, Jessica ; Grout, Jason ; Corlay, Sylvain ; Ivanov, Paul ; Avila, Damián ; Abdalla, Safia ; Willing, Carol: Jupyter Notebooks – a publishing format for reproducible computational workflows. In: Loizides, F. (Hrsg.) ; Schmidt, B. (Hrsg.): *Positioning and Power in Academic Publishing: Players, Agents and Agendas* IOS Press (Veranst.), 2016, S. 87 – 90

[Kondratyuk and Straka 2019] Kondratyuk, Dan ; Straka, Milan: 75 Languages, 1 Model: Parsing Universal Dependencies Universally. In: *EMNLP-IJCNLP 2019 - 2019 Conf. Empir. Methods Nat. Lang. Process. 9th Int. Jt. Conf. Nat. Lang. Process. Proc. Conf.* (2019), apr, S. 2779–2795. – URL http://arxiv.org/abs/1904.02099

[Kyselý 2017] Kyselý, Radek: *kysely/sentiment-analysis-czech: Conducting and publishing sentiment analysis experiments in the Czech language*. 2017. – URL `https://github.com/kysely/sentiment-analysis-czech`. – Zugriffsdatum: 2020-11-17

[Lan et al. 2019] Lan, Zhenzhong ; Chen, Mingda ; Goodman, Sebastian ; Gimpel, Kevin ; Sharma, Piyush ; Soricut, Radu: ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In: *arXiv* (2019), sep. – URL `http://arxiv.org/abs/1909.11942`

[Lenc and Hercig 2016] Lenc, Ladislav ; Hercig, Tomás: Neural Networks for Sentiment Analysis in Czech. In: Brejová, Broňa (Hrsg.): *Proceedings of the 16th ITAT: Slovenskočeský NLP workshop (SloNLP 2016)* Bd. 1649. Bratislava, Slovakia : CreateSpace Independent Publishing Platform, 2016, S. 48–55. – ISBN 978-1537016740

[Lewis et al. 2019] Lewis, Mike ; Liu, Yinhan ; Goyal, Naman ; Ghazvininejad, Marjan ; Mohamed, Abdelrahman ; Levy, Omer ; Stoyanov, Ves ; Zettlemoyer, Luke: BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In: *arXiv* (2019), oct. – URL `http://arxiv.org/abs/1910.13461`

[Libovický et al. 2018] Libovický, Jindřich ; Rosa, Rudolf ; Helcl, Jindřich ; Popel, Martin: Solving Three Czech NLP Tasks End-to-End with Neural Models. In: *Proc. 18th Conf. ITAT 2018 Slov. NLP Work. (SloNLP 2018)*, CreateSpace Independent Publishing Platform, Košice, 2018, S. 138–143. – URL `https://www.yelp.com/dataset/`. – ISBN 11234/12839

[Lin et al. 2017] Lin, Zhouhan ; Feng, Minwei ; Santos, Cicero N. dos ; Yu, Mo ; Xiang, Bing ; Zhou, Bowen ; Bengio, Yoshua: A Structured Self-attentive Sentence Embedding. In: *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.* (2017), mar. – URL `https://arxiv.org/abs/1703.03130v1`

[Ling et al. 2016] Ling, Wang ; Luís, Tiago ; Marujo, Luís ; Fernandez, Ramón ; Amir, Astudillo S. ; Dyer, Chris ; Black, Alan W. ; Trancoso, Isabel: Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation. In: *arXiv* (2016)

[Liu et al. 2020] Liu, Qi ; Kusner, Matt J. ; Blunsom, Phil: A Survey on Contextual Embeddings. In: *arXiv* (2020), mar. – URL `http://arxiv.org/abs/2003.07278`

[Liu et al. 2019] Liu, Yinhan ; Ott, Myle ; Goyal, Naman ; Du, Jingfei ; Joshi, Mandar ; Chen, Danqi ; Levy, Omer ; Lewis, Mike ; Zettlemoyer, Luke ; Stoyanov, Veselin: RoBERTa: A Robustly Optimized BERT Pretraining Approach. In: *arXiv* (2019), jul. – URL `http://arxiv.org/abs/1907.11692`

[Malaviya et al. 2019] Malaviya, Chaitanya ; Wu, Shijie ; Cotterell, Ryan: A Simple Joint Model for Improved Contextual Neural Lemmatization.

In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota : Association for Computational Linguistics, Juni 2019, S. 1517–1528. – URL https://aclanthology.org/N19-1155

[Marcus et al. 1993] Marcus, Mitchell ; Santorini, Beatrice ; Marcinkiewicz, Mary A.: Building a Large Annotated Corpus of English: The Penn Treebank. In: *Tech. Reports* (1993), oct. – URL https://repository.upenn.edu/cis{_}reports/237

[McCann et al. 2017] McCann, Bryan ; Bradbury, James ; Xiong, Caiming ; Socher, Richard: Learned in translation: Contextualized word vectors. In: *Adv. Neural Inf. Process. Syst.* Bd. 2017-Decem, 2017. – ISSN 10495258

[McCulloch et al. 1943] McCulloch, WS ; Biophysics, W Pitts T. bulletin of mathematical ; 1943, Undefined: A logical calculus of the ideas immanent in nervous activity. In: *Bull. Math. Biophys. 5, 115–133* (1943). – URL https://link.springer.com/article/10.1007{%}252FBF02478259

[Michel et al. 2019] Michel, Paul ; Levy, Omer ; Neubig, Graham: Are Sixteen Heads Really Better than One? In: *arXiv* (2019), may. – URL http://arxiv.org/abs/1905.10650

[Mikolov et al. 2013] Mikolov, Tomas ; Chen, Kai ; Corrado, Greg ; Dean, Jeffrey: Distributed Representations of Words and Phrases and their Compositionality. URL http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and, 2013. – Forschungsbericht

[Minsky and Papert 2017] Minsky, M ; Papert, SA: Perceptrons: An introduction to computational geometry. (2017). – URL https://www.google.com/books?hl=cs{&}lr={&}id=PLQ5DwAAQBAJ{&}oi=fnd{&}pg=PR5{&}dq=perceptrons+an+introduction+to+computational+geometry{&}ots=zzCzAMspY0{&}sig=x0HLubdLNN3Irw4cZUlmdyHK8BM

[Mitchell 1997] Mitchell, Tom M.: *Machine Learning*. McGraw-Hil. New York : McGraw-Hill, 1997. – 99 S

[Montoyo et al. 2012] Montoyo, Andrés ; Martínez-Barco, Patricio ; Balahur, Alexandra: Subjectivity and sentiment analysis: An overview of the current state of the area and envisaged developments. In: *Decis. Support Syst.* Bd. 53, nov 2012, S. 675–679. – ISSN 01679236

[Müller et al. 2015] Müller, Thomas ; Cotterell, Ryan ; Fraser, Alexander ; Schütze, Hinrich: Joint lemmatization and morphological tagging with lemming. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, S. 2268–2274

[Oakley et al. 1995] Oakley, Brian et al.: *Final Evaluation Of The Results Of Eurotra: A Specific Programme Concerning the preparation of the development of an operational EUROTRA system for Machine Translation*. 1995

[Pan and Yang 2010] Pan, Sinno J. ; Yang, Qiang: A Survey on Transfer Learning. In: *IEEE Trans. Knowl. Data Eng.* 22 (2010), Nr. 10, S. 1345–1359. – URL http://socrates.acadiau.ca/courses/comp/dsilver/NIPS95

[Paszke et al. 2019] Paszke, Adam ; Gross, Sam ; Massa, Francisco ; Lerer, Adam ; Bradbury, James ; Chanan, Gregory ; Killeen, Trevor ; Lin, Zeming ; Gimelshein, Natalia ; Antiga, Luca ; Desmaison, Alban ; Kopf, Andreas ; Yang, Edward ; DeVito, Zachary ; Raison, Martin ; Tejani, Alykhan ; Chilamkurthy, Sasank ; Steiner, Benoit ; Fang, Lu ; Bai, Junjie ; Chintala, Soumith: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H. (Hrsg.) ; Larochelle, H. (Hrsg.) ; Beygelzimer, A. (Hrsg.) ; Alché-Buc, F. dś (Hrsg.) ; Fox, E. (Hrsg.) ; Garnett, R. (Hrsg.): *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, S. 8024–8035

[Pedregosa et al. 2011] Pedregosa, F. ; Varoquaux, G. ; Gramfort, A. ; Michel, V. ; Thirion, B. ; Grisel, O. ; Blondel, M. ; Prettenhofer, P. ; Weiss, R. ; Dubourg, V. ; Vanderplas, J. ; Passos, A. ; Cournapeau, D. ; Brucher, M. ; Perrot, M. ; Duchesnay, E.: Scikit-learn: Machine Learning in Python. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830

[Pennington et al. 2014] Pennington, Jeffrey ; Socher, Richard ; Manning, Christopher D.: GloVe: Global Vectors for Word Representation. URL http://nlp., 2014. – Forschungsbericht

[Peters et al. 2017] Peters, Matthew E. ; Ammar, Waleed ; Bhagavatula, Chandra ; Power, Russell: Semi-supervised sequence tagging with bidirectional language models. In: *ACL 2017 - 55th Annu. Meet. Assoc. Comput. Linguist. Proc. Conf. (Long Pap.* Bd. 1, 2017

[Peters et al. 2018] Peters, Matthew E. ; Neumann, Mark ; Iyyer, Mohit ; Gardner, Matt ; Clark, Christopher ; Lee, Kenton ; Zettlemoyer, Luke: Deep contextualized word representations. In: *NAACL HLT 2018 - 2018 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf.* Bd. 1, Association for Computational Linguistics (ACL), feb 2018, S. 2227–2237. – URL http://allennlp.org/elmo. – ISBN 9781948087278

[Plisson et al. 2004] Plisson, Joël ; Lavrac, Nada ; Mladenic, Dunja: A Rule based Approach to Word Lemmatization. In: *Proc. 7th Int. Multiconference Inf. Soc.*, 2004, S. 83–86

[Pota et al. 2021] Pota, Marco ; Ventura, Mirko ; Catelli, Rosario ; Esposito, Massimo: An effective BERT-based pipeline for Twitter sentiment analysis: a case study in Italian. In: *Sensors* 21 (2021), Nr. 1, S. 133

[Putra et al. 2020] Putra, Ilham F. ; Purwarianti, Ayu ; Ai-Vlb, U-Coe: Improving Indonesian Text Classification Using Multilingual Language Model. In: *Int. Conf. Adv. Informatics Concept, Theory Appl.* (2020). – URL https://www.yelp.com/dataset

[Radford Alec et al. 2019] Radford Alec ; Wu Jeffrey ; Child Rewon ; Luan David ; Amodei Dario ; Sutskever Ilya: Language Models are Unsupervised Multitask Learners — Enhanced Reader. In: *OpenAI Blog* 1 (2019), Nr. 8

[Radfort et al. 2018] Radfort, Alec ; Narasimhan, Karthik ; Salimans, Tim ; Sutskever, Ilya: Improving Language Understanding by Generative Pre-Training. In: *OpenAI* (2018)

[Raffel et al. 2019a] Raffel, Colin ; Shazeer, Noam ; Roberts, Adam ; Lee, Katherine ; Narang, Sharan ; Matena, Michael ; Zhou, Yanqi ; Li, Wei ; Liu, Peter J.: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. In: *arXiv* 21 (2019), oct, S. 1–67. – URL `http://arxiv.org/abs/1910.10683`

[Raffel et al. 2019b] Raffel, Colin ; Shazeer, Noam ; Roberts, Adam ; Lee, Katherine ; Narang, Sharan ; Matena, Michael ; Zhou, Yanqi ; Li, Wei ; Liu, Peter J.: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. In: *arXiv* 21 (2019), oct, S. 1–67. – URL `http://arxiv.org/abs/1910.10683`

[Ramachandran et al. 2017] Ramachandran, Prajit ; Liu, Peter J. ; Le, Quoc V.: Unsupervised pretraining for sequence to sequence learning. In: *EMNLP 2017 - Conf. Empir. Methods Nat. Lang. Process. Proc.*, Association for Computational Linguistics (ACL), 2017, S. 383–391. – ISBN 9781945626838

[Reback et al. 2020] Reback, Jeff ; jbrockmendel ; McKinney, Wes ; Bossche, Joris V. den ; Augspurger, Tom ; Cloud, Phillip ; Hawkins, Simon ; gfyoung ; Sinhrks ; Roeschke, Matthew ; Klein, Adam ; Petersen, Terji ; Tratner, Jeff ; She, Chang ; Ayd, William ; Hoefler, Patrick ; Naveh, Shahar ; Garcia, Marc ; Schendel, Jeremy ; Hayden, Andy ; Saxton, Daniel ; Gorelli, Marco E. ; Shadrach, Richard ; Jancauskas, Vytautas ; McMaster, Ali ; Li, Fangchen ; Battiston, Pietro ; Seabold, Skipper ; attack68 ; Dong, Kaiqi: *pandas-dev/pandas: Pandas*. Februar 2020. – URL `https://doi.org/10.5281/zenodo.3509134`

[Rogers et al. 2020] Rogers, Anna ; Kovaleva, Olga ; Rumshisky, Anna: A Primer in BERTology: What we know about how BERT works. In: *arXiv* (2020), feb. – URL `http://arxiv.org/abs/2002.12327`

[Rosa and Mareček 2019] Rosa, Rudolf ; Mareček, David: Inducing Syntactic Trees from BERT Representations. In: *arXiv* (2019), jun. – URL `http://arxiv.org/abs/1906.11511`

[Rosenblatt 1958] Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. In: *Psychol. Rev.* 65 (1958), nov, Nr. 6, S. 386–408. – ISSN 0033295X

[Ruder et al. 2019] Ruder, Sebastian ; Peters, Matthew E. ; Swayamdipta, Swabha ; Wolf, Thomas: Neural Transfer Learning for Natural Language Processing. In: *Proc. 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Tutorials*, 2019, S. 15–18

[Rumelhart et al. 1986] Rumelhart, DE ; Hinton, GE ; Nature, RJ W. ; 1986, Undefined: Learning representations by back-propagating errors. URL https://www.nature.com/articles/323533a0, 1986. – Forschungsbericht

[Russakovsky et al. 2015] Russakovsky, Olga ; Deng, Jia ; Su, Hao ; Krause, Jonathan ; Satheesh, Sanjeev ; Ma, Sean ; Huang, Zhiheng ; Karpathy, Andrej ; Khosla, Aditya ; Bernstein, Michael ; Berg, Alexander C. ; Fei-Fei, Li: ImageNet Large Scale Visual Recognition Challenge. In: *Int. J. Comput. Vis.* 115 (2015), dec, Nr. 3, S. 211–252. – URL https://link.springer.com/article/10.1007/s11263-015-0816-y. – ISSN 15731405

[Russell et al. 1995] Russell, Stuart J. ; Norvig, Peter ; Canny, John F. ; Malik, Jitendra M. ; Edwards, Douglas D.: Artificial Intelligence A Modern Approach. 1995. – Forschungsbericht. – 529 S. – ISBN 0131038052

[dos Santos et al. 2016] Santos, Cicero dos ; Tan, Ming ; Xiang, Bing ; Zhou, Bowen: Attentive Pooling Networks. URL http://arxiv.org/abs/1602.03609, feb 2016. – Forschungsbericht

[Schmid 1995] Schmid, Helmut: Improvements In Part-of-Speech Tagging With an Application To German. In: *In Proceedings of the ACL SIGDAT-Workshop*, 1995, S. 47–50

[Schwenk et al. 2006] Schwenk, Holger ; Dchelotte, Daniel ; Gauvain, Jean-Luc: Continuous Space Language Models for Statistical Machine Translation. 2006. – Forschungsbericht. – 723–730 S

[Sido et al. 2021] Sido, Jakub ; Pražák, Ondřej ; Přibáň, Pavel ; Pašek, Jan ; Seják, Michal ; Konopík, Miloslav: Czert – Czech BERT-like Model for Language Representation. (2021), mar. – URL https://arxiv.org/abs/2103.13031v2

[Stickland and Murray 2019] Stickland, Asa C. ; Murray, Iain: BERT and PALs: Projected attention layers for efficient adaptation in multi-task learning. In: *36th Int. Conf. Mach. Learn. ICML 2019* Bd. 2019-June, 2019

[Straka 2018] Straka, Milan: UDPipe 2.0 Prototype at CoNLL 2018 UD Shared Task. In: *Proc. CoNLL 2018 Shar. Task Multiling. Parsing from Raw Text to Univers. Depend.* (2018), S. 197–207. – URL http://ufal.mff.cuni.cz/udpipe.

[Straka et al. 2021] Straka, Milan ; Náplava, Jakub ; Straková, Jana ; Samuel, David: RobeCzech: Czech RoBERTa, a monolingual contextualized language representation model. (2021), may. – URL http://arxiv.org/abs/2105.11314

[Straka et al. 2019a] Straka, Milan ; Straková, Jana ; Hajič, Jan: Czech Text Processing with Contextual Embeddings: POS Tagging, Lemmatization, Parsing and NER. In: *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)* 11697 LNAI (2019), sep, S. 137–150. – URL http://arxiv.org/abs/1909.03544

[Straka et al. 2019b] Straka, Milan ; Straková, Jana ; Hajič, Jan: Evaluating Contextualized Embeddings on 54 Languages in POS Tagging, Lemmatization and Dependency Parsing. In: *arXiv* (2019), aug. – URL http://arxiv.org/abs/1908.07448

[Straka et al. 2019c] Straka, Milan ; Straková, Jana ; Hajič, Jan: Regularization with Morphological Categories, Corpora Merging. URL https://github.com/google-research/, 2019. – Forschungsbericht. – 95–103 S

[Straková et al. 2014] Straková, Jana ; Straka, Milan ; Hajič, Jan: Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition. URL http://ufal.mff.cuni.cz/morce/index.php, 2014. – Forschungsbericht. – 13–18 S

[Sun et al. 2019] Sun, Chi ; Qiu, Xipeng ; Xu, Yige ; Huang, Xuanjing: How to Fine-Tune BERT for Text Classification? In: *Chinese Comput. Linguist.* Springer International Publishing, 2019, S. 194–206. – URL https://github.

[Sutskever et al. 2014] Sutskever, Ilya ; Vinyals, Oriol ; Le, Quoc V.: Sequence to Sequence Learning with Neural Networks. In: *Adv. Neural Inf. Process. Syst.* 4 (2014), sep, Nr. January, S. 3104–3112. – URL http://arxiv.org/abs/1409.3215

[Szegedy et al. 2015] Szegedy, Christian ; Liu, Wei ; Jia, Yangqing ; Sermanet, Pierre ; Reed, Scott ; Anguelov, Dragomir ; Erhan, Dumitru ; Vanhoucke, Vincent ; Rabinovich, Andrew: Going Deeper with Convolutions. URL https://www.cv-foundation.org/openaccess/content{_}cvpr{_}2015/html/Szegedy{_}Going{_}Deeper{_}With{_}2015{_}CVPR{_}paper.html, 2015. – Forschungsbericht

[Taylor 1953] Taylor, Wilson L.: "Cloze Procedure": A New Tool for Measuring Readability. In: *Journal. Q.* 30 (1953), sep, Nr. 4, S. 415–433. – URL http://journals.sagepub.com/doi/10.1177/107769905303000401. – ISSN 0022-5533

[Tenney et al. 2019a] Tenney, Ian ; Das, Dipanjan ; Pavlick, Ellie: BERT Rediscovers the Classical NLP Pipeline. In: *ACL 2019 - 57th Annu. Meet. Assoc. Comput. Linguist. Proc. Conf.* (2019), may, S. 4593–4601. – URL http://arxiv.org/abs/1905.05950

[Tenney et al. 2019b] Tenney, Ian ; Xia, Patrick ; Chen, Berlin ; Wang, Alex ; Poliak, Adam ; McCoy, R T. ; Kim, Najoung ; Van Durme, Benjamin ; Bowman, Samuel R. ; Das, Dipanjan ; Pavlick, Ellie: What do you learn from context? Probing for sentence structure in contextualized word representations. In: *arXiv* (2019), may. – URL http://arxiv.org/abs/1905.06316

[Tibshirani 1996] Tibshirani, Robert: Regression Shrinkage and Selection Via the Lasso. In: *J. R. Stat. Soc. Ser. B* 58 (1996), jan, Nr. 1, S. 267–288. – URL https://rss.onlinelibrary.wiley.com/doi/full/10.1111/j.

2517-6161.1996.tb02080.xhttps://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1996.tb02080.xhttps://rss.onlinelibrary.wiley.com/doi/10.1111/j.2517-6161.1996.tb02080.x. – ISSN 0035-9246

[Turian et al. 2010] Turian, Joseph ; Ratinov, Lev ; Bengio, Yoshua: Word representations: A simple and general method for semi-supervised learning. Association for Computational Linguistics, 2010. – Forschungsbericht. – 11–16 S. – URL http://metaoptimize.

[Vaswani et al. 2017] Vaswani, Ashish ; Brain, Google ; Shazeer, Noam ; Parmar, Niki ; Uszkoreit, Jakob ; Jones, Llion ; Gomez, Aidan N. ; Kaiser, Łukasz ; Polosukhin, Illia: Attention Is All You Need. 2017. – Forschungsbericht

[Veselovská 2017] Veselovská, Kateřina: *SENTIMENT ANALYSIS IN CZECH*. 2017. – ISBN 9788088132035

[Virtanen et al. 2019] Virtanen, Antti ; Kanerva, Jenna ; Ilo, Rami ; Luoma, Jouni ; Luotolahti, Juhani ; Salakoski, Tapio ; Ginter, Filip ; Pyysalo, Sampo: Multilingual is not enough: BERT for Finnish. In: *arXiv* (2019), dec. – URL http://arxiv.org/abs/1912.07076

[Wagner and Fischer 1974] Wagner, Robert A. ; Fischer, Michael J.: The String-to-String Correction Problem. In: *J. ACM* 21 (1974), jan, Nr. 1, S. 168–173. – URL https://doi.org/10.1145/321796.321811. – ISSN 0004-5411

[Wilks 2005] Wilks, Yorick: The History of Natural Language Processing and Machine Translation. 2005. – Forschungsbericht

[Wolf et al. 2019] Wolf, Thomas ; Debut, Lysandre ; Sanh, Victor ; Chaumond, Julien ; Delangue, Clement ; Moi, Anthony ; Cistac, Pierric ; Rault, Tim ; Louf, R'emi ; Funtowicz, Morgan ; Brew, Jamie: HuggingFace's Transformers: State-of-the-art Natural Language Processing. In: *ArXiv* abs/1910.03771 (2019)

[Wu et al. 2016] Wu, Yonghui ; Schuster, Mike ; Chen, Zhifeng ; Le, Quoc V. ; Norouzi, Mohammad ; Macherey, Wolfgang ; Krikun, Maxim ; Cao, Yuan ; Gao, Qin ; Macherey, Klaus ; Klingner, Jeff ; Shah, Apurva ; Johnson, Melvin ; Liu, Xiaobing ; Kaiser, Łukasz ; Gouws, Stephan ; Kato, Yoshikiyo ; Kudo, Taku ; Kazawa, Hideto ; Stevens, Keith ; Kurian, George ; Patil, Nishant ; Wang, Wei ; Young, Cliff ; Smith, Jason ; Riesa, Jason ; Rudnick, Alex ; Vinyals, Oriol ; Corrado, Greg ; Hughes, Macduff ; Dean, Jeffrey: Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. In: *arXiv* (2016), sep. – URL http://arxiv.org/abs/1609.08144

[Yang et al. 2020] Yang, Cheng ; Wang, Shengnan ; Yang, Chao ; Li, Yuechuan ; He, Ru ; Zhang, Jingqiao: Progressively Stacking 2.0: A Multi-stage Layerwise Training Method for BERT Training Speedup. In: *arXiv* (2020), nov. – URL http://arxiv.org/abs/2011.13635

[Yang and Zhao 2019] Yang, Junjie ; Zhao, Hai: Deepening Hidden Representations from Pre-trained Language Models. In: *arXiv* (2019), nov. – URL http://arxiv.org/abs/1911.01940

[Yang et al. 2019a] Yang, Zhilin ; Dai, Zihang ; Yang, Yiming ; Carbonell, Jaime ; Salakhutdinov, Ruslan ; Le, Quoc V.: XLNet: Generalized Autoregressive Pretraining for Language Understanding. In: *arXiv* (2019), jun. – URL http://arxiv.org/abs/1906.08237

[Yang et al. 2019b] Yang, Zhilin ; Dai, Zihang ; Yang, Yiming ; Carbonell, Jaime ; Salakhutdinov, Ruslan ; Le, Quoc V.: XLNet: Generalized Autoregressive Pretraining for Language Understanding. In: *arXiv* (2019), jun. – URL http://arxiv.org/abs/1906.08237

[Yildiz and Tantuğ 2019] Yildiz, Eray ; Tantuğ, A. C.: Morpheus: A Neural Network for Jointly Learning Contextual Lemmatization and Morphological Tagging, 2019

[Yosinski et al. 2014] Yosinski, Jason ; Clune, Jeff ; Bengio, Yoshua ; Lipson, Hod: How transferable are features in deep neural networks? In: *Adv. Neural Inf. Process. Syst.* 4 (2014), nov, Nr. January, S. 3320–3328. – URL http://arxiv.org/abs/1411.1792

[Zhang et al. 2019] Zhang, Zhengyan ; Han, Xu ; Liu, Zhiyuan ; Jiang, Xin ; Sun, Maosong ; Liu, Qun: ERNIE: Enhanced Language Representation with Informative Entities. In: *ACL 2019 - 57th Annu. Meet. Assoc. Comput. Linguist. Proc. Conf.* (2019), may, S. 1441–1451. – URL http://arxiv.org/abs/1905.07129

[Žižka and Dařena 2010] Žižka, Jan ; Dařena, František: Automatic Sentiment Analysis Using the Textual Pattern Content Similarity in Natural Language. In: *Int. Conf. Text, Speech Dialogue*, Springer, 2010

List of Figures

List of Tables

A. Attachements

A.1 Learning Rate for Tagging and Lemmatization

A	40:1e-3,20:1e-4
B	40:1e-3,20:1e-4
C	60:1e-3
D	40:1e-3,20:1e-4,2:2e-5
E	60:1e-3,5:3e-5
F	20:2e-5
G	20:3e-5

Table A.1: Legend for table 2.4.

A.2 The Most Frequent Tags Improved by the Best (tl_18) Model

Freq	Tag	Freq	Tag
163	NNNXX-----A----	27	AAMS1----1A----
151	NNIS1-----A----	25	NNMP4-----A----
130	NNIS4-----A----	22	NNNP1-----A----
122	NNNS4-----A----	22	NNFS6-----A----
103	NNFP1-----A----	21	NNFXX-----A---8
99	NNMS1-----A----	20	NNNS6-----A----
84	NNFP4-----A----	19	PDNS1----------
83	NNNS1-----A----	19	J,----------
72	NNMS4-----A----	19	AAIP4----1A----
69	AAIS4----1A----	18	NNNS3-----A----
68	NNFS1-----A----	18	AAFS4----1A----
64	AAFP1----1A----	17	AAFS1----1A----
62	AAIS1----1A----	16	PDNS4----------
60	NNFS4-----A----	16	NNNP4-----A----
58	RR--4----------	16	NNIS3-----A----
56	NNFS2-----A----	15	AANP1----1A----
54	RR--6----------	14	J^----------
48	NNNS2-----A----	13	NNIP7-----A----
48	NNMS2-----A----	13	NNFP2-----A----
47	VB-P---3P-AA---	13	Cn-S1----------
47	NNIP1-----A----	12	P7-X4----------
47	AANS4----1A----	12	NNIS2-----A----
43	Db----------	12	AAMP2----1A----
42	NNIP4-----A----	11	RR--2----------
38	VB-S---3P-AA---	11	NNMS3-----A----
38	NNFS3-----A----	11	AGFP1-----A----
35	AANS1----1A----	11	AAXXX----1A----
35	AAFS2----1A----	11	AAMS2----1A----
35	AAFP4----1A----	11	AAFS3----1A----
33	AAIP1----1A----	10	NNFXX-----A----
28	NNFS7-----A----	10	AAMP4----1A----

Table A.2: Most frequent tags, which were correctly predicted by *tl_18*, but incorrectly by *tl_1* (=baseline).

A.3 The Most Frequent Tags – the Best Model vs mBERT

Freq	Tag	Freq	Tag
70	NNIS1-----A----	23	Db----------
69	NNNS4-----A----	23	AAFS2----1A----
55	NNIS4-----A----	22	RR--6----------
55	NNFP1-----A----	22	NNIP1-----A----
45	NNNS1-----A----	21	NNFS3-----A----
45	NNFS1-----A----	20	AANS1----1A----
38	NNMS4-----A----	16	AAIP1----1A----
38	NNFP4-----A----	15	NNNS3-----A----
37	AAFP1----1A----	15	NNFXX-----A---8
36	NNFS2-----A----	15	NNFS7-----A----
34	NNFS4-----A----	15	AAFP4----1A----
32	VB-S---3P-AA---	13	PDNS1----------
30	AAIS1----1A----	13	NNIS3-----A----
29	VB-P---3P-AA---	13	AAIP4----1A----
28	RR--4----------	12	PDNS4----------
27	NNNXX-----A----	11	NNNS6-----A----
27	NNMS2-----A----	11	AAFS4----1A----
27	AAIS4----1A----	10	NNMP4-----A----
26	NNNS2-----A----	10	J,----------
26	AANS4----1A----	10	AAXXX----1A----
25	NNMS1-----A----	10	AAMS1----1A----
23	NNIP4-----A----		

Table A.3: Most frequent tags, which were correctly predicted by *tl_18*, but incorrectly by *tl_3* (= best model with mBERT).