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

Dedication.

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:

Keywords: Natural Language Processing BERT Bidirectional Encoder Representations from Transformers

Contents

List of Abbreviations

AI	Artificial Intelligence. 3
NLP	Natural Language Processing. 1, 3, 4, 8
	. 3
SOTA	state-of-the-art. 3

Introduction

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.

The work is divided into seven chapters. First chapter presents theoretical background in NLP and used Artificial Intelligence (AI) methods. This quite general chapter is followed by a thorough description of implemented tasks, each in its own separate chapter (Lemmatization and part-of-speech tagging, Sentiment analysis, Language modelling). Description of each task includes its definition, previous work and state-of-the-art results, methods applied in this work and their results. Implementation details like code overview, third-party libraries, technical problems and their solution can be found in chapter 2. For personal examination and exploration of presented models serves user documentation of attached models in chapter 5 and text is closed by a discussion about results (chapter 6) and a conclusion with 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 NLP

Natural language processing 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. Its focus 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 a grammatical categories of the given word. Both of these types of tasks are important for NLP.

Morphological analysis tasks used in this work are lemmatization and 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 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* and the lemma is *jablko*.

POS tagging

POS tagging classifies word into one of the POS categories (like a noun, pronoun, or verb) (Hladká). As a part of tagging task can sometimes be considered a determination of grammatical categories (e.g., case, number, or tense).

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 the 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 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 can be involved so-called subjectivity(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.”. The subjectivity of the claim does not depend on its sentiment. This part of sentiment analysis was not included in this work, and it is mentioned just for completeness.

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 are co-called Brown corpus (Francis and Kucera, 1979) – first electronic corpus mixed from newspaper articles and fiction literature, and PennTreebank (Marcus et al., 1993), which is the first syntactically annotated corpus but has quite a domain limited source - articles from Wall Street Journal. The internal structure of corpora can be of many types. 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

[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.

picture 1.2. The 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’s 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 (like tweets on US Airlines pages [3] or movie reviews [4]) Moreover, they are annotated for one type of task (in the case of example datasets it is sentiment analysis).

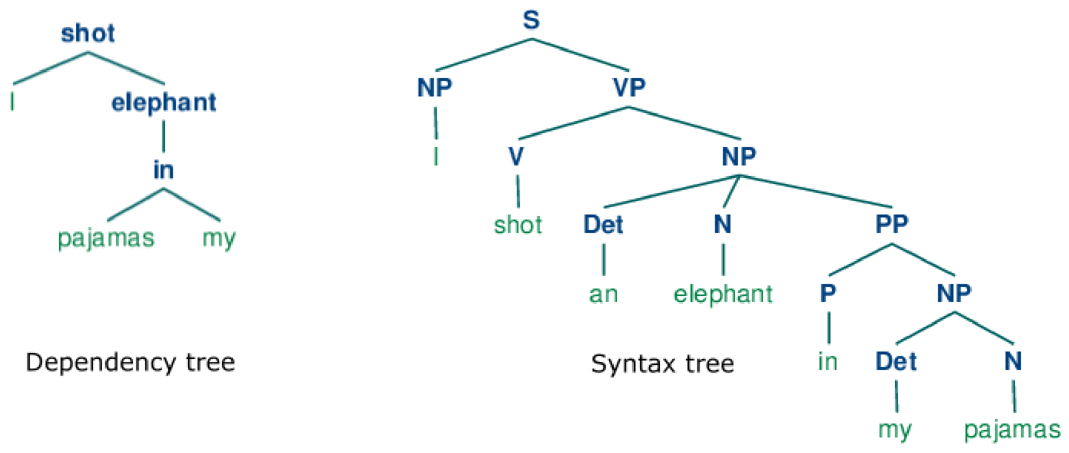


Figure 1.1: An example of the syntax and dependency tree. The dependency tree, as the name indicates, describes dependencies between words. Such dependencies are of various types; for example, an elephant in the example text is a direct object of the shooting action. A root of such a tree is typically 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 machine translation, 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 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

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

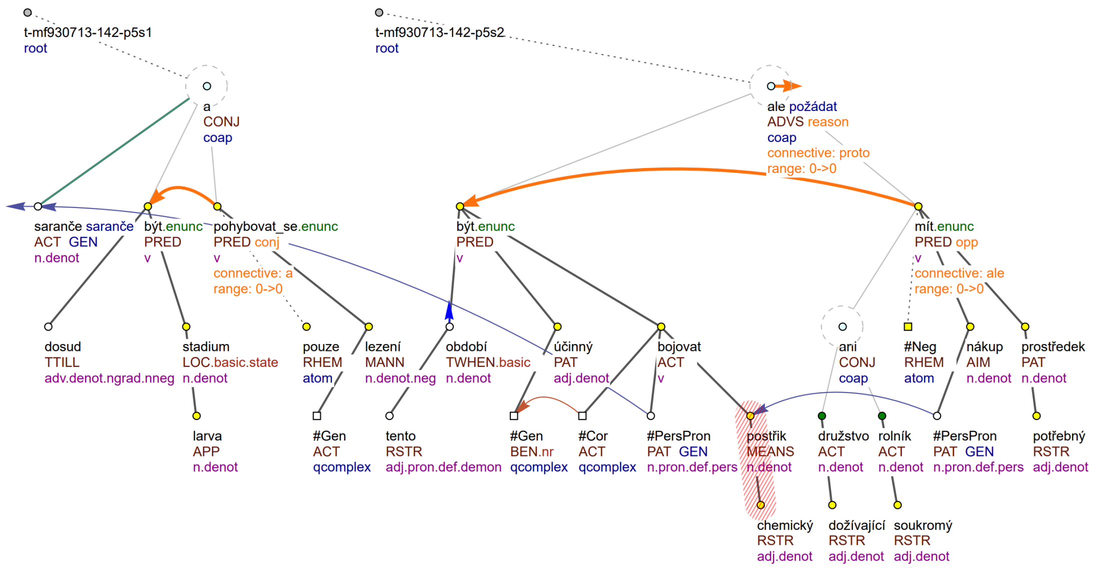


Figure 1.2: Prague dependency treebank example PDT35 for the sentences: *Grasshoppers are still in the larvae stadium, crawling only. At this time of the year, it is efficient to fight them using chemicals, but neither the ailing cooperatives nor private farmers can afford them.* czech: *Sarančata jsou 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. This example is from Prague dependency treebank, which 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.

people share their opinions and emotions and are ready to make their choices according to incoming influences, social networks are an important source of information. The big problem while analyzing this data is their amount. Twitter users alone 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 also 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 their sequences (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 (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),(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 obtained even better results (a neural network learns probabilities of words) (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). 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 [6]. For that reason, the following subsection presents deep learning basics, their usage, and improvements in NLP in recent years.

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

1.2 Deep Learning

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.). 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 Fig. 1.3). A perceptron's input is a vector of features describing an input example, and output is classification into class 0 or 1.

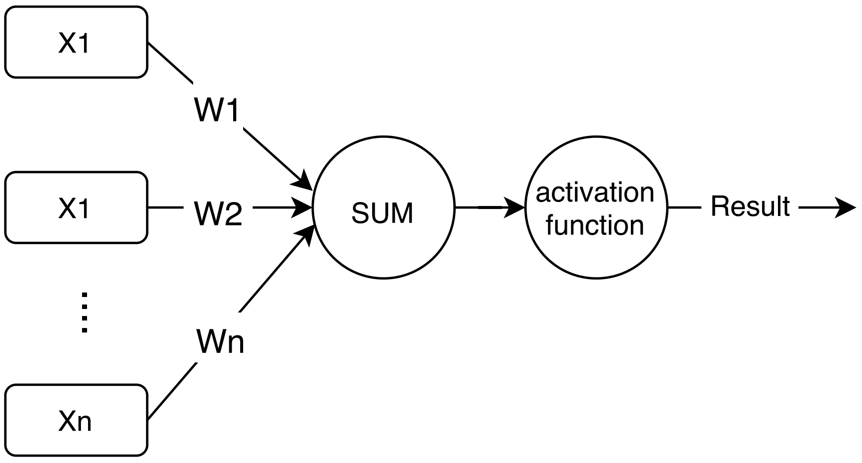


Figure 1.3: A one-layer perceptron architecture. The result is formed by the application of the activation function on a weighted sum of inputs. Weights are updated during training till it returns satisfactory results.

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

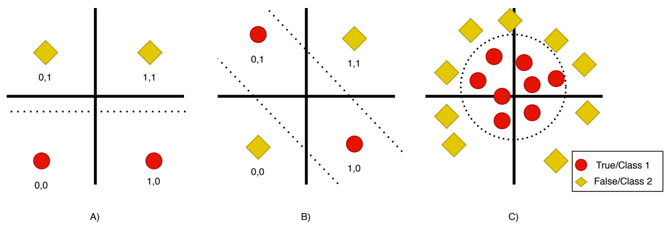


Figure 1.4: This picture illustrates XOR problem. Perception can find the correct solution only if the data are linearly separable. It means that they can be divided by a hyperplane. An example of such two-dimensional data can be seen in picture A). The dotted line shows a possible border for separation. 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 a circle (pic. 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 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 DNN is a multilayer perceptron (see picture 1.5). It is combined with 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 number and shape of layers, activation functions, and connections between neurons. (see Fig. 1.5).

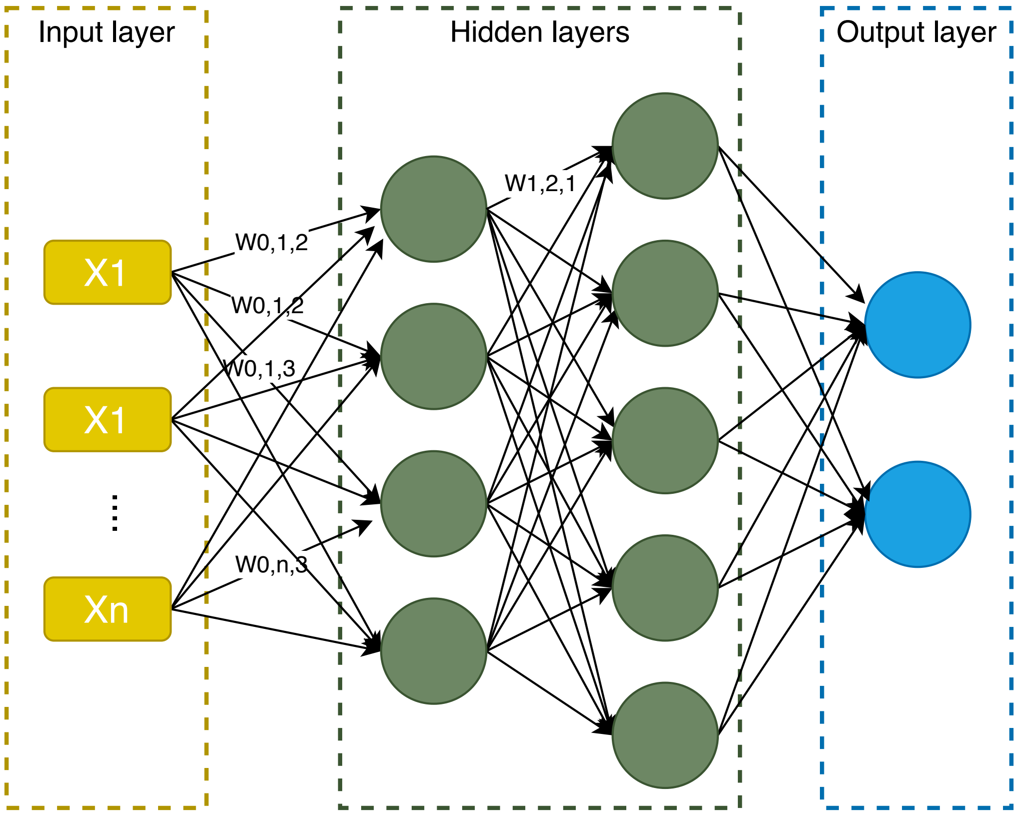


Figure 1.5: Multilayer perceptron (or feed-forward neural network) is formed of input and output layer and a variable number of hidden layers with different sizes. In every layer, the chosen application function is applied to a weighted sum of inputs from the previous layer.

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 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[7]. 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 sensless. 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 itself. 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. 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 them in the required amount because they are usually created manually by humans [8]. 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 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. 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,

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

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

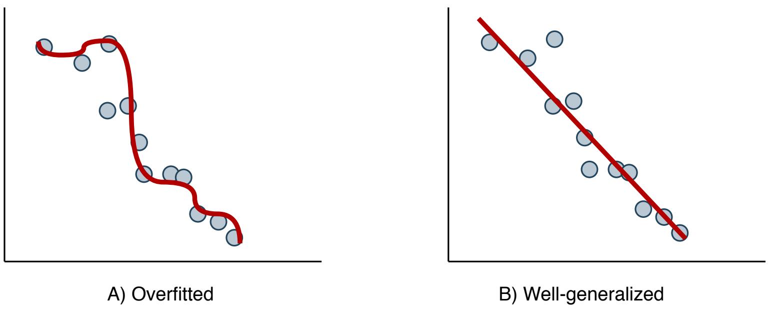


Figure 1.6: Figure A) presents overfitting scenario. Figure B) illustrates possible well-generalized solution.

therefore is not limited to an NLP only. In any 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 [9]. Instead of this, label smoothing applies following formula:

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

where K is number of classes, n is a number of examples, 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:

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

$p(y \mid x_i)$ being the truth labels' distribution and $q_\theta(y_i \mid x_i)$ being the predicted distribution of labels. After substitution of label smoothing[10]:

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

for $u(y \mid x_i)$ uniform distribution (can be replaced by $1/K$) which gaves after multiplication [11]:

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

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.

[9]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.

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

[11]see footnote 10.

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 count 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 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: CBOW and SkipGram model. The objective of CBOW model is to predict a masked 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 $number\_of\_words \times embedding\_size$. 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 few nearest words, but the statistics in the dataset are not explicitly used. GloVe (Pennington et al., 2014) embeddings, on the other hand, uses 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., 2019c), (Liu et al., 2020).

Contextualized embeddings

Contextualized embeddings were invented in recent years, namely BERT (Devlin et al., 2019), ELMo (Peters et al., 2018) 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 compound of them. Section 1.2.7 describes the possibilities of involving and training such embeddings in more detail. 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 Attention mechanism

Attention mechanism (Bahdanau et al.) 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 the words, which represents this connections (see picture 1.7).

When the task is a 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 picture 1.8 An improvement to a 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.

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

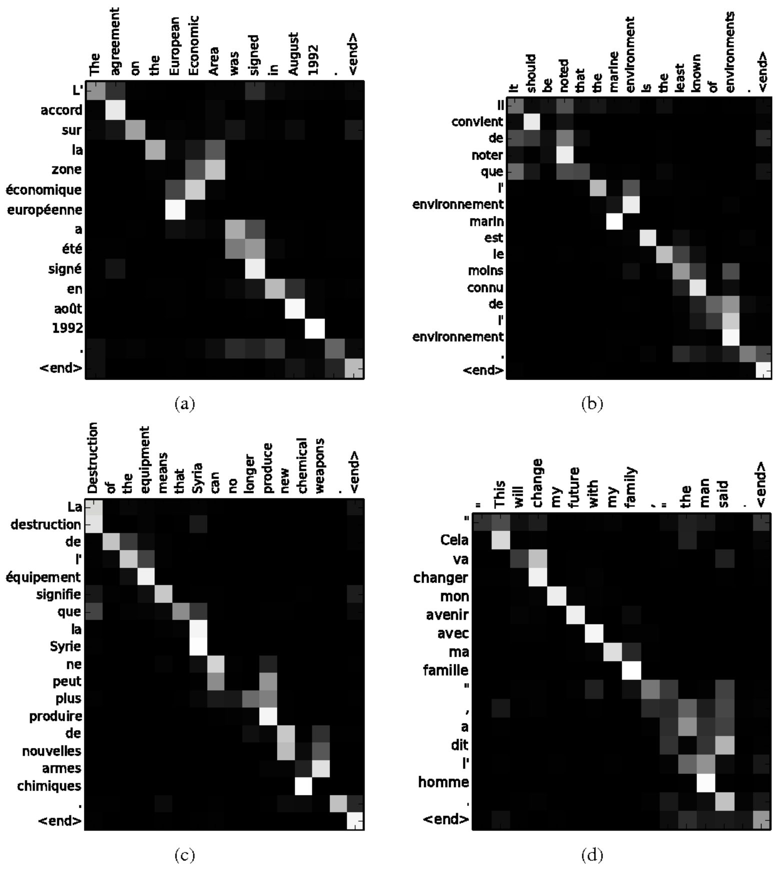


Figure 1.7: Figure 3 of (Bahdanau et al.) presents the use of the attention for machine translation. Axis show words of a sentences in English and French. The importance for a translation between a pair of words is represented by a lightness (more lighter = more important).

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 do not 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 picture 1.10).

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

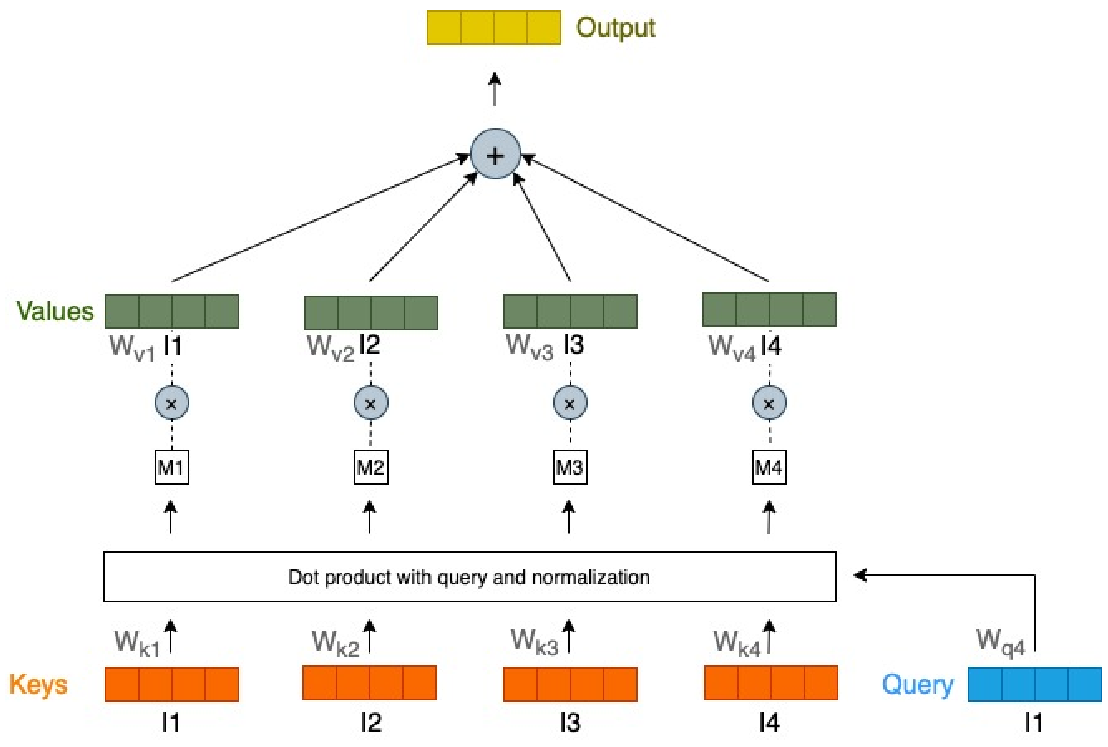


Figure 1.8: Self-attention mechanism scheme for one selected query vector. The result is an embedding, which is improved by the context of the word. This picture illustrates the result for the embedding of the first word (l1) in a four-word long text. Input words are denoted as l1 to l4. Keys , values, and a query are all multiplied 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 a query are computed. A 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.

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 gates). Comparison of both architectures can be found in figure 1.11.

LSTM

LSTM cell uses three gates: input, output and forget gate. Every gate is composed by sigmoid function with actual input and 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

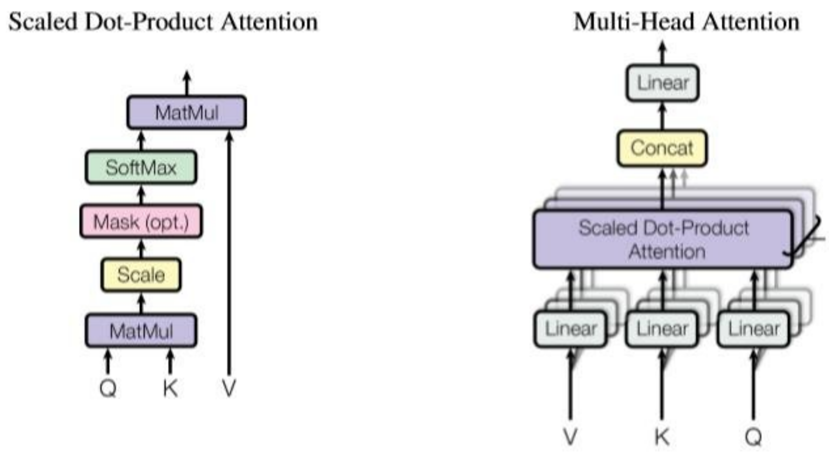


Figure 1.9: Figure 2 from (Vaswani et al., 2017) describes the attention mechanism used in Transformers. Scaled Dot-Product Attention, as authors call it, is basically the same architecture as described in the previous figure (1.8). The dimension of values (d_v) is different from query and keys dimensions (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.

1.12.

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. Update gate serves as a weight for a combination of previous and current states, which form the new output. For more detailes see Figure 1.13.

1.2.6 Transformers

The transformers solve the same problem as RNNs but proposes a different architecture. A 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's basic idea is the following: 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

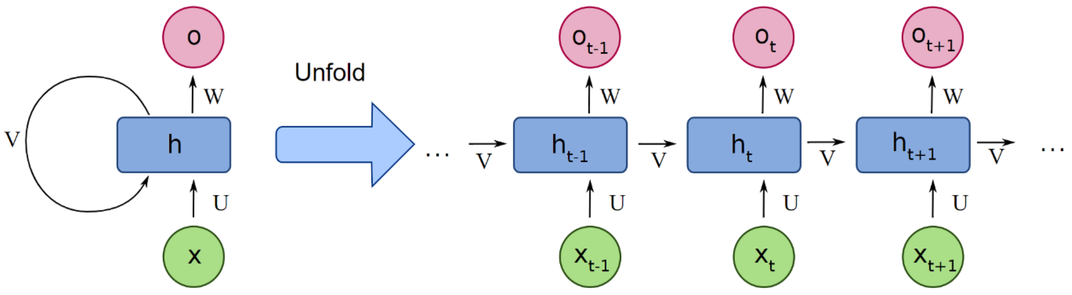


Figure 1.10: Basic Recurrent neural network architecture is showed on the left side of the figure. It is composed by one rnn cell which recurrently uses informations from previously seen input. X is the input of the cell, e.g. a word, O is the output for the given word (e.g. its translation or next word 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.*

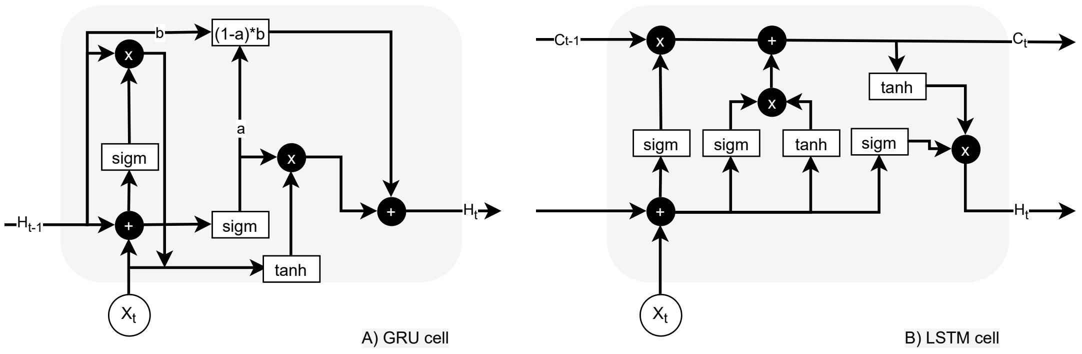


Figure 1.11: Comparison of LSTM and GRU architecture. x_t is the current input (current word), Source: *http://dprogrammer.org/rnn-lstm-gru.*

sentence in the target language with the same meaning as an original input (see figures 1.14 and 1.15).

A 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 previous words (there are $-inf$ values in the positions on the right side). For a representation of word position in the 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.

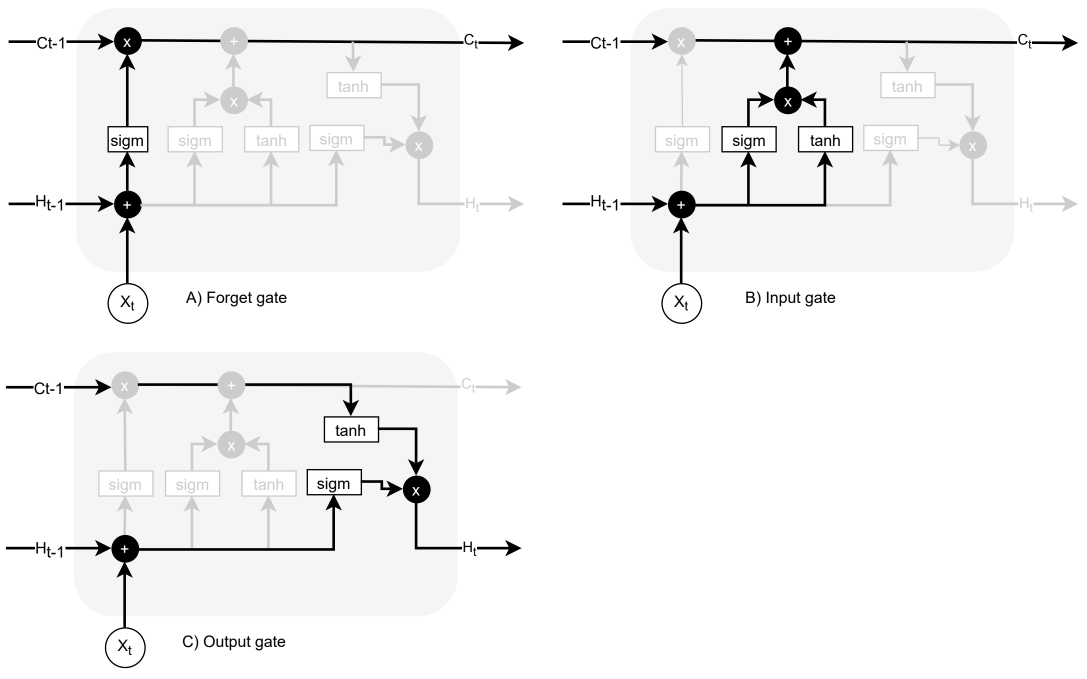


Figure 1.12: An architecture of the LSTM cell, epsecially its gating mechanism. A LSTM cell carry through the time, in addition to a hidden state, also a cell state, which serves as a long-term memory. *A forget gate* is responsible for choosing the amount of cell state, therefore the information from 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.

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 one thousand classes. Many large deep models were trained on this dataset and then applied to different computer vision tasks with great success (Huh et al.).

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. For natural language processing, transfer learning is currently mainly represented by contextualized embeddings

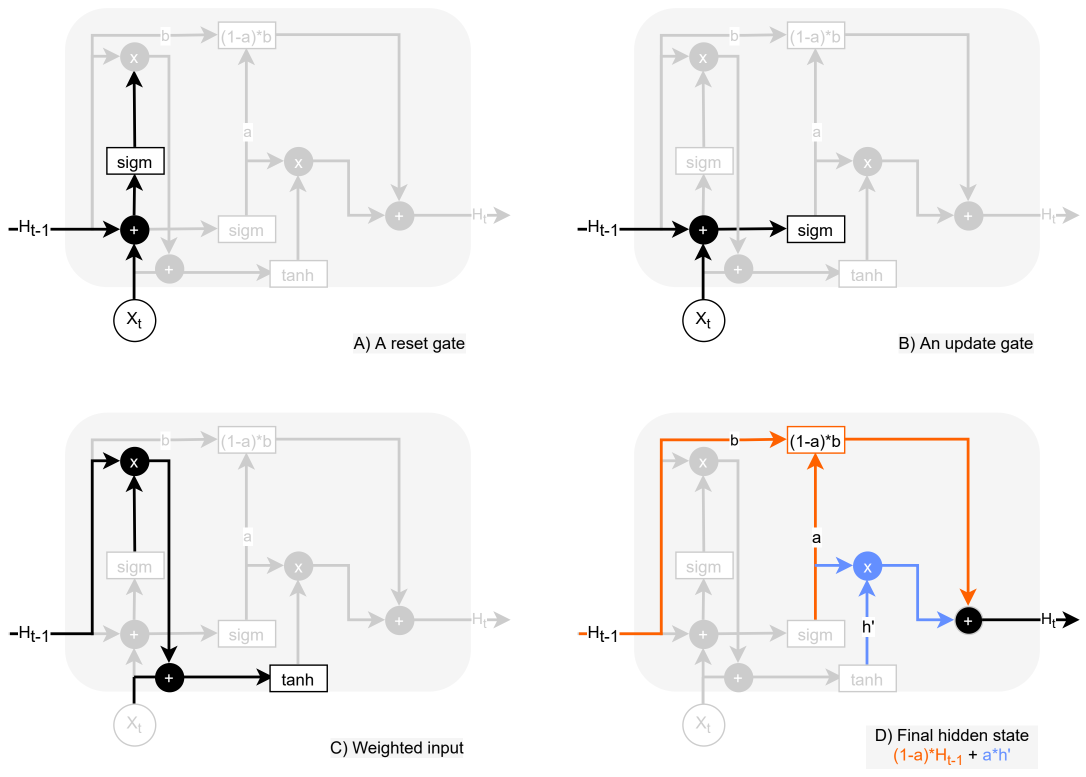


Figure 1.13: An architecture of GRU cell. *An update gate (A)* computes how much of previous information should be passed to next cell based on previous hidden state and the current time input. *A reset gate (B)* uses the same formula (although the input and the previous hidden state have different weights) 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.

obtained from pretrained language models. Contextualization of embeddings is 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[12]. Supervised methods use machine translation, which is the classic NLP task, but also natural language inference or other tasks with the potential to capture general knowledge about language.

[12]This is a slight terminology inaccuracy. In section 1.2.2, unsupervised learning was defined as a task where data does not contain 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 $t_n, N \in \mathbb{N}$ 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 and decode 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 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 *finetune* 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 is offered in 1.3.3.

1.3 BERT and its descendants

The methods described in subsection are utilized in the BERT-like models, that 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, showed better ability to capture the language knowledge and meant a turning point 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. biLSTM are bidirectional LSTMs. 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 labeled datasets, where is almost always not enough data). (Peters et al., 2017) uses 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 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 an input is a 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 presented 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 with 48 layers. Each layer is a transformer decoder with the 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 BERT with the depth comparable to GPT 2, but using a different part of transformers. It uses modified pre-training tasks compared to other classical language modeling and has a context from both sides.This work mainly relies on Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019) and its variants, so this section describes and explains the main features of this model. BERT is a pre-trained language model that is finetuned for many other tasks, so it is an example of transfer learning (see 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 [13]. Training of the language model requires a considerable amount of data and computational time, 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 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 Transformers architecture,

which 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.

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 and is designed to minimize unknown 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 lost 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 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.

[13]Some published models, which can be used in popular python library Tranformers from HugginFace[14], 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*.

	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).

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 first. In BERT is no recurrent cell, and instead, position embeddings 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 different from original transformers, where the position was also encoded as embeddings, but embeddings were fixed, not learned.

Segment embeddings These embeddings indicate whether the token belongs to the first or second part of the input. It has 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 second part.

Architecture

BERT adapts encoder part of architecture from 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 finished with one fully connected layer for a specific task (see Fig. 1.19).

Original paper proposes two main architecture hyperparameters version, base and large (see table 1.1), in the dependence on the number of layers (L), 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 modeling (MLM). These two tasks were chosen because BERT's authors believe they can 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 text. A goal of the task is whether sentence B is following or random, 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 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 80% chance to be replaced by a MASK token, 10% chance to be replaced by a random token, or it will remain unchanged with 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 MASK token 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 methods:

- 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 stated by BERT.
- An extension of applicability – BERT model works nice 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 will be 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 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 will be 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 will 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) fall into the third category of BERT family. Great achievements 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 more than BERT, which has 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 (2K 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 $BERT_{large}$, but it combined bidirectional BERT training with objectives strengthening usage of a left context, which is important for language generation, and the model also uses NSP. The following rule selects training objectives: 1/3 of time, MLM objective (same as BERT) is used, 1/3 of time 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 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, 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 original or replaced token. Therefore, the model has two parts – generator and discriminator (loosely on spored by Generative Adversial Networks). It significantly decreases training time because data are used more efficiently. Even with shorter training time, ELELCTRA outperforms both XLNet and RoBERTa.

T5 (= ”Text-to-Text Transfer Transformer”) (Raffel et al., 2019a) is another approach to language generating and also another deep study of various parameters' influence. The training objective is similar to BERT; they choose 15% of tokens and replace them with the mask token. One change is presented – it showed to be useful to select whole spans of the text of length three and replace them with one mask token rather than randomly select 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 is in the text form and also every output. 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 (like in GPT). The main benefit of 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 were proven to be the best, although the performance of various pre-training objectives varied for different tasks. The first objective changes the position of sentences. Text infilling replaces a randomly chosen span of text with 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 the (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: involves the large model as a teacher and a smaller model (possibly with completely different architecture) as a student. Stu-

dent 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 decrease the vocabulary size of embeddings.

This study showed a possibility of reducing BERT to the quarter of 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 to this, ALBERT presents a modification to NSP objective: sentence-order prediction (SOP).

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) on BERT layers, but whit would be time and memory more consuming. This method is valid, and the results will more likely be improved by using BERT embeddings over some other types. Even the BERT paper study shows that using a concatenation of the last four layers can achieve comparable performance to finetuned BERT, but this method still seems to lose some potential of language models compared to learning the model on specific task (Sun et al.).

Speaking about fine-tuning, there are many decision to be made, i.e., how to chose learning rate, whether to train all layers together or freeze some of them, if its needed 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 are known, and main choices with possibilities will be offered here.

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, for sequence classification is mostly used CLS token, but using 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, although it seems to have no 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). Often possibility is a combination of the last four layers (the best option in (Sun et al.) was max). Generally spoken, usage of more layers proved to be advantageous, and it is possible to let the choice of layers 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 on data for the task domain or at least in the task data language as it showed to be beneficial (Sun et al.). More technical details to be decided are following:

Classification Part Architecture 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 (finetuned) 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 the small learning rate, the goal is to customize the language model a bit for current tasks. 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 $3e^{-6}$ and $5e^{-5}$ (Devlin et al., 2019), (Virtanen et al., 2019), (Kittask et al., 2020) as the higher learning rate can cause a catastrophic forgetting (Sun et al.). 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 the good results (Howard and Ruder, 2018), (Sun et al.), (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 successful training.

Last, there is also possible to select between different optimizer 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 the faster learning. BERT proved to be nicely scalable in selecting the batch size, and simply the bigger batch size usually works better (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 the model knows 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 showed 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. A for the attention heads, many papers show that original architecture with sixteen-headed multi-head attention is no 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 one head only 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 author also shows 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).

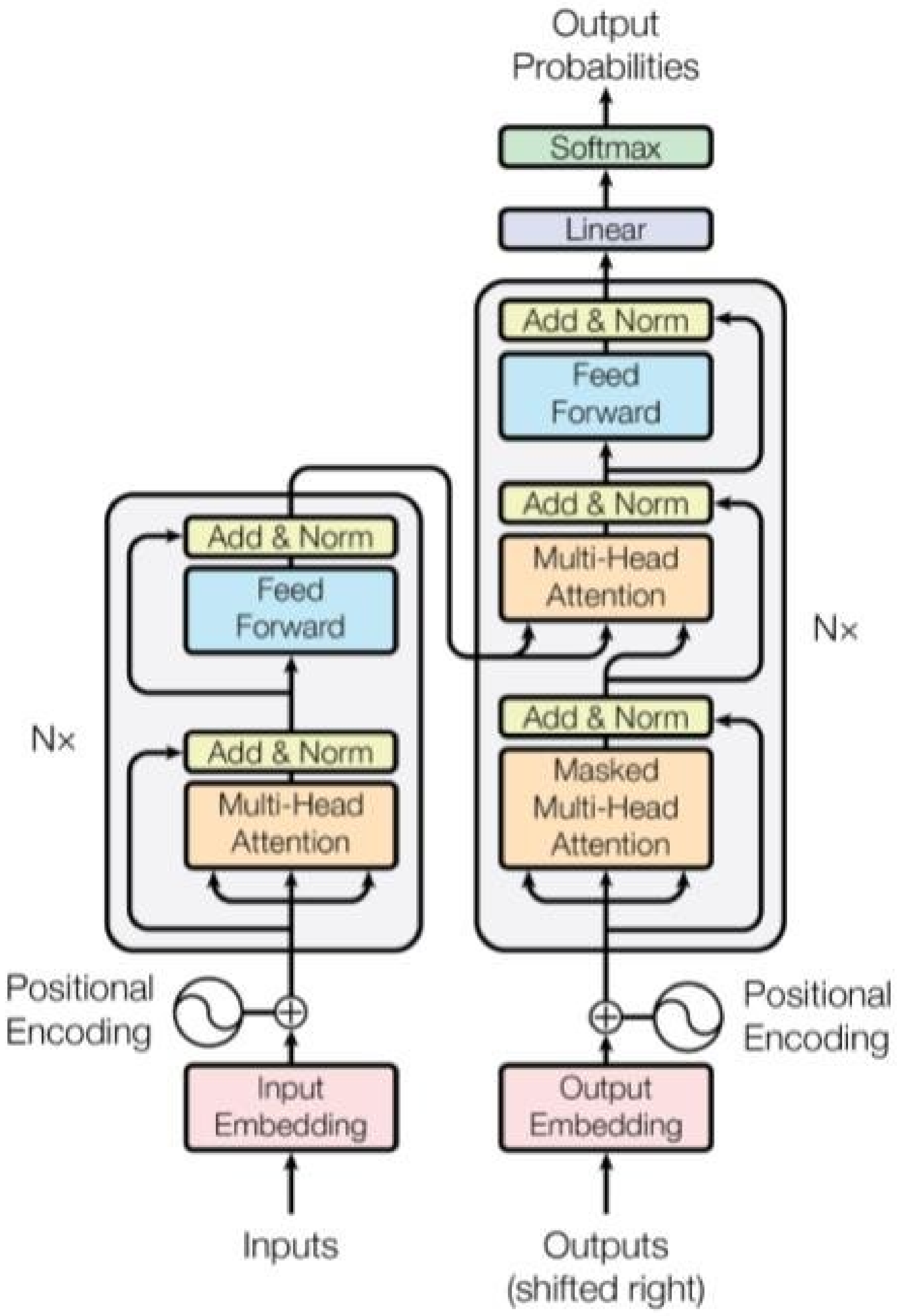


Figure 1.14: This picture describes design of one encoder-decoder block in detail. Every encoder layer consist of self-attention and feed-forward layer supplied with normalization and residual connections. Source: http://jalammar.github.io/illustrated-transformer

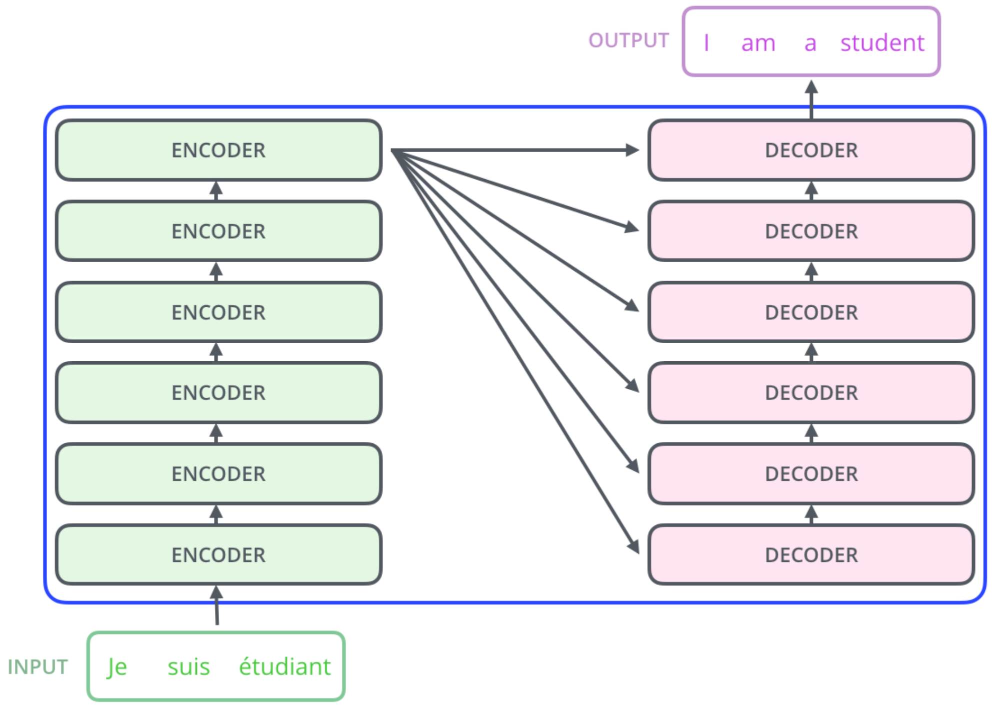


Figure 1.15: In transformers, encoder and decoder parts are both composed by many of block of respective types. The input goes first through a series of encoders and than the output of encoder part is put into every decoder in the decoder part. source: http://jalammar.github.io/illustrated-transformer/

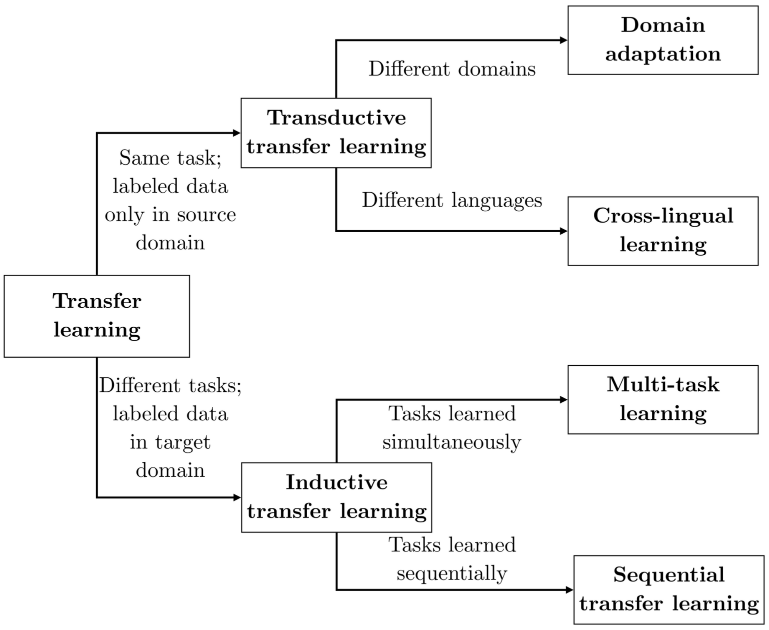

Domain adaptation
Different domains
Transductive transfer learning
Same task; labeled data only in source domain
Different languages
Cross-lingual learning
Transfer learning
Different tasks; labeled data in target domain
Multi-task learning
Tasks learned simultaneously
Inductive transfer learning
Tasks learned sequentially
Sequential transfer learning

Figure 1.16: Figure from (Ruder et al., 2019) offers possible taxonomy for transfer learning. Following definiton in (Pan and Yang, 2009), 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).

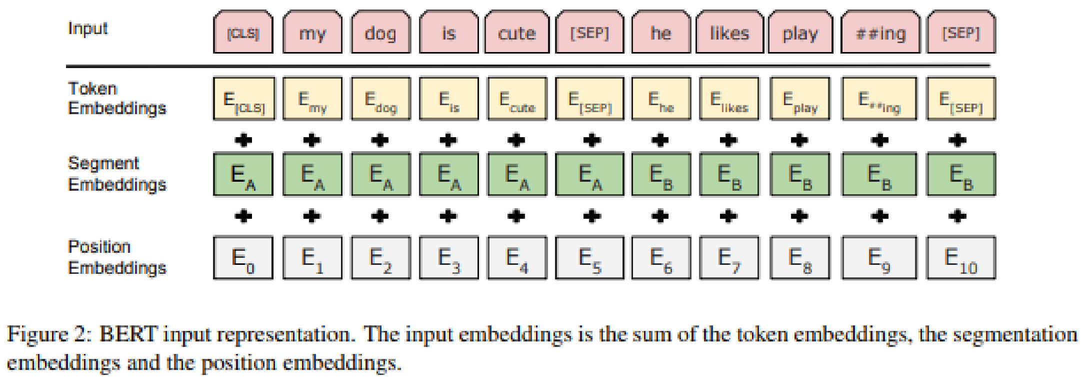

Input
[CLS] my dog is cute [SEP] he likes play ##ing [SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}
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)*.

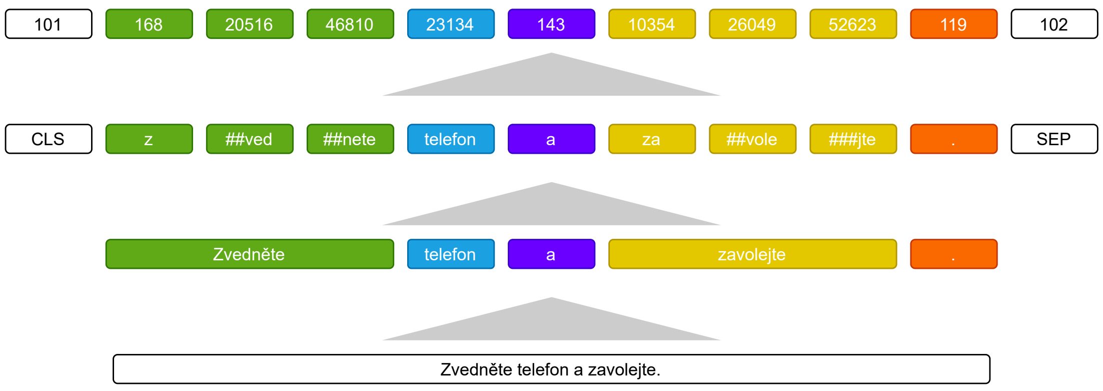


Figure 1.18: This figure illustrates a transformation of one input sentence (from PDT3) to suit bert input expectations. The sentence is divided into words and then into tokens from wordpiece tokenizer vocabulary. Accents may be removed depending on the used model. The sentence is decorated with special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

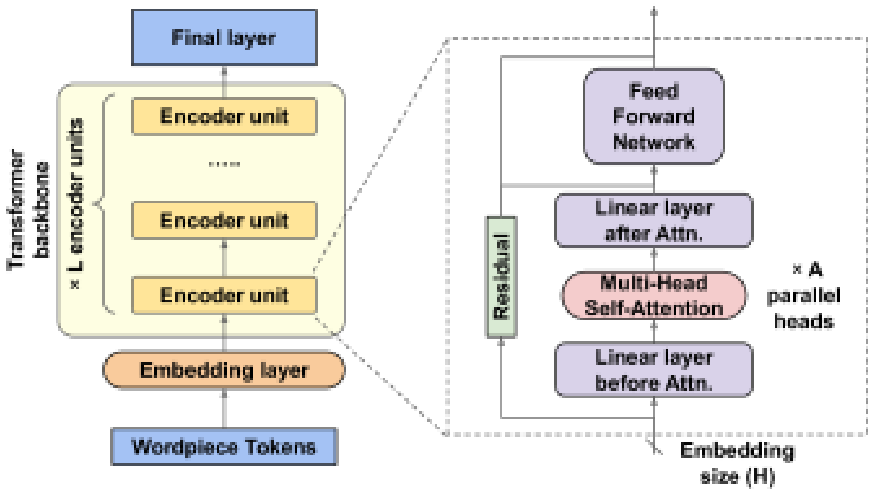


Figure 1.19: A figure from (Ganesh et al., 2020) describes BERT architecture in detail.

2. Task 1: Lemmatization and part-of-speech tagging

Czech morphology developement is dated from 1989 and in description of words uses 15-places morphological tags (Hana and Zeman, 2005). A more detailed description of each position can be seen in table 2.

This work aims to improve previously published SOTA results for contextualized embeddings in czech lemmatization and tagging (Straka et al., 2019a).

(Horsmann and Zesch) (Plank et al.) (Plisson et al.) (Straka et al., 2019d) (Straka et al., 2019c) (Toutanova et al., 2003) (Wang et al., 2015) (Huang et al., 2015) (Collobert et al., 2011) ... preprocessing

The practical part of this work is focused on these tasks:

- POS tagging
  input: a word
  output: part-of-speech tags – as noun, pronoun, punctuation mark etc.

- lemmatization
  input: a word
  output: lemma – a base form of a given words, meaning for example nominative of singular for nouns or infinitive for verbs.

- sentiment analysis
  input: a sentence or a sequence of sentences
  output: prevailing sentiment of the input from categories: neutral, positive, negative.

doplnit diskuzi o ruzných možnostech definice

. This chapter describes implemented linguistic models. As mentioned before, this work implements models for three Czech NLP tasks: tagging, lemmatization and sentiment analysis. Common model for first two tasks develops on the paper by (Straka et al., 2019b) focused on application of contextual embeddings produced by language models as Bert or Flair. Third task, sentiment analysis, is performed by adding only one fully-connected layer at the top of bert model. So in the opposite to previous tasks, no sophisticated handcrafted pipeline is built and model relies only on the network powers of language structure representation.

2.1 POS tagging and lemmatization model

The model for this part is build upon a model (and a code) for previous work on Czech NLP processing with contextual embeddings (Straka et al., 2019b). Data preparation pipeline - tokenization and sentence segmentation is taken over from the paper as well as base structure of lemmatizer and tagger network.

2.1.1 Dataset and preprocessing

Dataset for these tasks is based on Prague Dependency Treebank (PDT), specifically version 3.5, which is PDT edition from year 2018. Dataset is divided into

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: Table from (Hana and Zeman, 2005) describes all 15 positions of Czech morphological tagging.

Experiment	Without Dictionary			With Dictionary		
	Tags	Lemmas	Both	Tags	Lemmas	Both
StrakaB	97.94%	98.75%	97.31%	98.05%	98.98%	97.65%
emb (lr 0.0001)	97.80%	98.70%	97.17%	97.95%	98.93%	97.55%
baseline	97.04 %	98.56 %	96.41%	97.31 %	98.83 %	96.90%
StrakaC	97.67%	98.63%	97.02%	97.91%	98.94%	97.51%

Table 2.2: Straka2019B is the best solution from (Straka et al., 2019a) paper. Straka2019C is a comparable solution (BERT embeddings only), which was transformed into TF2 in this work.

three parts - train, dtest and etest. Second one is used as development set while the third one as a test set for validation. Data consists of sentences with lemmas and tags.Input sentences are preprocessed as follows:

- white space deletion
- mapping characters – all unknown dev and test characters are then mapped into one *unk* token.
- mapping words from train into integers – all unknown dev and test words are then mapped into one *unk* token.

2.1.2 Network

Lemmatization as well as tagging are considered to be classification tasks (same as in (Straka et al., 2019b)), i.e. lemmatization classification target are lemma generating rules and tagging target is tag, of course. Both tasks share one network.

Script allows training of these model variants:

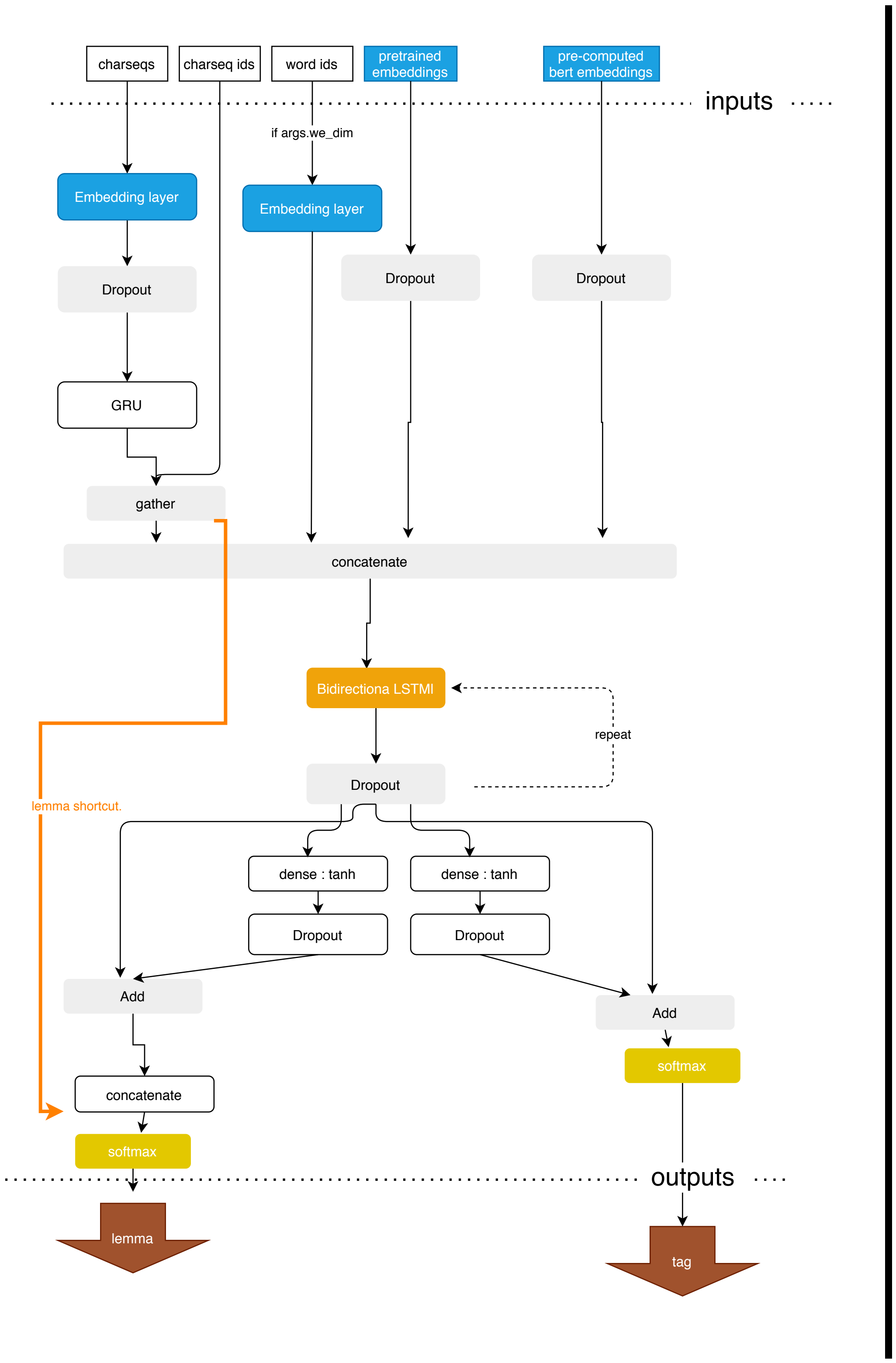
- baseline model – original implementation of network as described in figure **Embeddings:** Model contains potentially embeddings of four types – pre-trained word2vec embeddings , word embeddings trained with network, bert embeddings (just in some models as described later) and character-level embeddings.

- label smoothing – baseline model with label smoothing

- bert embeddings – model can take precomputed bert embeddings as an input, otherwise embeddings are computed at the beginning and also saved for further use. In this case, data are tokenized by BERT tokenizer. Same principle is used for unknown words as before. Embeddings for word segments are averaged for each original input word, as BERT word segments and sentence words does not correspond to each other one to one. These embeddings serves as additional input to the network as in figure

- BERT finetunning – In contrast to the BERT embeddings model, in this case BERT embeddings are also trained during network training, i.e. whole BERT network is chained to baseline network instead of embedding numbers only, gradients are backpropagated throw whole new network and weights are updated also in the BERT part of network.

All classification can be done with or without use of a morphological dictionary MorFlex . If the option with dictionary is used, generated tags and lemmas are chosen just from the dictionary. So it is selected lemma and/or tag with maximal likelihood, but just from those presented in the dictionary.



Metric used for evaluation of the model is accuracy.

3. Task 2: Sentiment Analysis

3.1 Task definition

As stated in (Veselovská): ”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, although there are some other definitions consisting of e.g. opinion extraction, irony or stance (Montoyo et al., 2012). Another tasks related to sentiment analysis is a subjectivity analysis (whether the presented opinion is objective or highly subjective), which is also not included in this work, mainly because the lack of labelled data for Czech. It is possible to analyze individual expressions, sentences or whole documents (Veselovská) and this work focuses on the document level classification.

3.2 Related Work

citace: (Çano and Bojar, 2019) (Kittask et al., 2020) .. estonian (Lenc and Hercig, 2016) sentiment in czech (Hercig et al., 2018) (Li et al.) (Libovický et al.) presents state-of-the art results in three czech NLP tasks including sentiment analysis. They use only CSFD dataset with resulting accuracy 80.8 .1 which is still not so good as Brychcín (81.5.3). The second mentioned paper, however, uses quite complicated method for classification incorporating the fact of which movie is reviewed.

Bachelor thesis which aplies bert on sentiment analysis in czech: results are accuracy 84 with tfidf and about 81 with bert and they used mall dataset.

3.3 Datasets

Four main Czech datasets with sentiment annotation are available - user reviews from MALL.cz, film reviews from csfd.cz, news from Aktualne.cz and posts from czech branch pages on facebook. As Aktualnecz dataset turned out to be problematical as the text were ambiguous for annotators, and its authors later used other mentioned datasets (Veselovská), this work also focuses only on the three other data sources – MALL, CSFD and Facebook [1]. Following part summarize each of used datasets.

3.3.1 Facebook

Length: 9752
From: ?
Labels: neutral, positive, negative

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

3.3.2 csfd

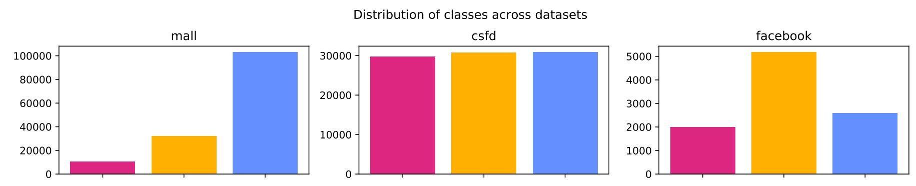
Length: 91304
From:
Labels: negative, positive ?

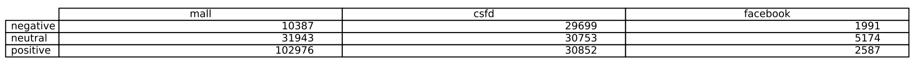
3.3.3 mallcz

Length: 145306
From:
Domain: domestic appliance reviews
Labels: negative, neutral, positive

3.3.4 imdb

Length: 25000
From: Labels: negative, positive





	mall	csfd	facebook
negative	10387	29699	1991
neutral	31943	30753	5174
positive	102976	30852	2587

Figure 3.1:

As can be seen in figure 3.1 distribution of labels differs among datasets. Moreover, Figure 3.2 shows, that the resulting dataset is highly unbalanced, which causes divergence and stuck in training. Due to the big part of labels being positive, many learning strategies just ended with predicting only *positive* class, i.e. 55% accuracy, so unfortunately learned nothing. This was the the case for following experiments settings:
learning rate of $2e - 3$ for 10 epochs. This ended with predicting only one class (positive). Some progress showed the followign approach:"2:5e-5,1:2e-5, unfreezed" for facebook data only (which are balanced). This results are above 71 for accuracy (Test accuracy: 0.726 F1 metrics: 0.7077061630227595 weighted).

3.3.5 Experiments

I tried another approach: Because the facebook dataset is balanced, I pretrained the model first on facebook and after reaching a decent accuracy, i further trained the model on the join data: for ten epochs with learning rate 5e-5 and an effecctive

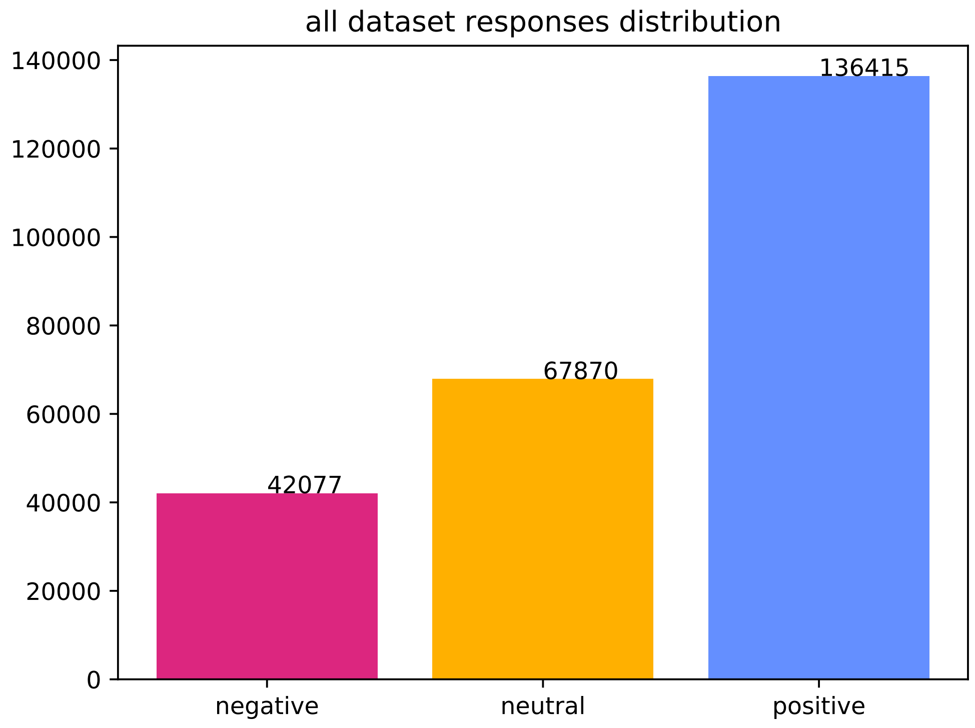


Figure 3.2:

batch size 48. [[6240 914 2783] [1236 4447 529] [1050 273 19020]] Test accuracy: 0.814068837005371 F1 metrics: 0.8087903170140932

The pretraining model was obtained as: without freezing, batch size 48, 2:5e-5,4:2e-5

Than I trained for 3 more epochs: [[6599 995 2343] [1246 4520 446] [1470 269 18604]] Test accuracy: 0.8145072892688808 F1 metrics: 0.8119431050948791 see 3.3

My baseline is tf-idf method, which resulted in [[4310 102 3015] [685 1016 779] [842 48 20227]] precision recall f1-score support

0 0.74 0.58 0.65 7427 1 0.87 0.41 0.56 2480 2 0.84 0.96 0.90 21117

accuracy 0.82 31024 macro avg 0.82 0.65 0.70 31024 weighted avg 0.82 0.82 0.81 31024

0.8236526560082517 acc,

Because BERT model was trained on multilingual data, it is naturally not so good in language minoritly presented in the Bert's training data. When transfering the learned knowledge to czech sentiment task, we actually want to improve model in two ways: teach it something more specific about given task, i.e., sentiment, and improve its knowledge about selected language (czech in this case). By using czech sentiment dataset, both thing are incorporated into training. To obtained better results and following the (Putra et al.), I also selected english sentiment dataset. The idea behind is that BERT is quite good in english and maybe can learn faster useful knowledge about the given task from data in more familiar language.

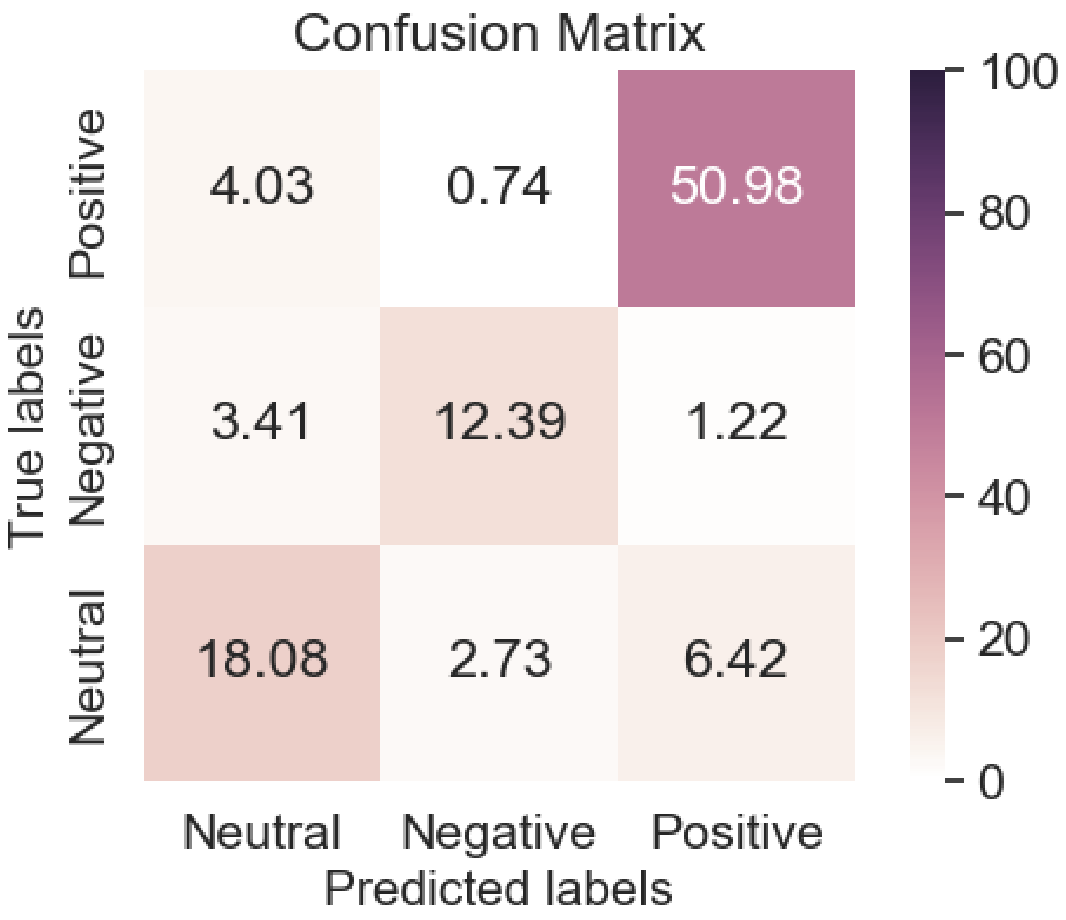


Figure 3.3: Normalized confusion matrix

4. 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 presented tasks. This chapter describes an implementation of all language models and other related code. It also presents all used libraries and technologies and this chapter is concluded with a presentation of experiment types common for training of both tasks presented in following chapters.

All code forms an attachment of this work and is also publicly available in GitHub [1].

4.1 Experiments

Firstly, all possibilities of experiment settings will be presented as the rest of the chapter serves for a specification of their implementation and running. Theoretical possibilities of training settings are described in detail in the previous section 1.3.3, but this part will describe particular training experiment settings, that are same for both downstream tasks [2].

4.1.1 General experiment settings (EXPE)

Training is performed in one of following settings:

- **base**: Baseline implementation (described separately for each task, typically without using advanced language models).
- **ls**: This uses same setting as baseline implementation but with label smoothing.
- **embed**: BERT-like language model is used only to generate static embeddings in advanced. These embeddings are not further trained.
- **fine**: Fine-tuning consist in dividing the training time into two parts. Firstly, rest of the model is trained with BERT layers frozen (not trained), so it is same as the *embed* settings. In the second part, the whole model is fine-tuned together.
- **simpl**e: Model architecture is reduced to bert layers with a simple classification head. This is basic setting for all sentiment analysis experiments but for tagging and lemmatization all previous choices, in contrast to this setting, are performed with more sophisticated classification head.
- **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 *simple* option).

[1]https://github.com/flower-go/DiplomaThesis

[2]Name in the bracket in titles and name of options refers to the name used in the tables of results and graphs.

4.1.2 Training data

Tagging and lemmatization tasks use same set of data for every experiments, co there is no need for separate description. Sentiment analysis task, however, uses three possible options for 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).

4.1.3 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 in advance different learning rates for different epochs.
- **isrd** *isrd* means inverse square root learning rate decay defined by formula: $1/\sqrt{max(n,k)}$ where k is the number of so-called *warmup steps* and n is the current iteration. This leads to constant learning rate for first k steps and decayed learning rate in rest of iterations.
- **cos**: Another learning rate scheduling used in this work is *cosine decay* which applies following formula:

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

  where lr^i_{min} and lr^i_{min} is the range of the learning rate, 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 the current epoch number.

4.1.4 Model layers selected for embeddings (LAYERS)

As discussed in the previous chapter, is unclear how to extract best embeddings from the language model, especially which layers to take into account. By selecting the most promising ways, following two hyperparameters are used in this work:

- **four**: Last four of the model are averaged to obtain final embeddings.
- **att**: This setting 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). Following section will explain technical details needed for running scripts.

4.2 Code description

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

4.2.1 Technologies description

All code is implemented in Python (v3.6.9). All dependencies and used libraries are listed in the requirements.txt file, but I should specifically mention some used libraries. 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.

Tensorflow and Keras

The main library used for developing 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 the code developed in Tensorflow.

Transformers

One of the most useful libraries in this work is Transformers library from Hugging Face (Wolf et al., 2019), which contains pretrained BERT models and tools for their usage as tokenizers.

Pandas

Pandas library (pandas development team, 2020) serves well for data analysis as it provides data structures like DataFrame, that 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 optimization.

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 an library which provides powerful multidimensional arrays with many predefined operations. It is fast and it is common to use it 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.

Artificial Intelligence Cluster

Other Resources

4.2.2 code structure

All code belonging to each of tasks is in the separate directory as can be seen on picture.

Runnable scripts

All code for **tagging and lemmatization** is placed in folder `morphodita_research`. Main files are `morpho_tagger_2.py` and `bert_finetunning_simple.py`, which serves for running all experiments relating to tagging and lemmatization. Script arguments are described in more detail in tables 4.2.2, 4.2.2 and 4.2.2.

Sentiment analysis experiments are runnable from `sentiment_analysis.py` with arguments as described in tables 4.2.2 and 4.2.2.

4.3 User guide

Argument	Values	Description
accu	int	Accumulation of gradient. Effective batch size is batch_size * accu.
batch_size	int	Batch size (without accumulation).
bert	string	Name of the bert model (from huggingFace library) or path to the model.
checkp	String	Name of the saved model weights. Saving weights is used instead of the whole model because of some technical issues.
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”	If not None, training will incorporate inverse square root decay or cosine decay for x episodes.
fine_lr	float	Different learning rate for the classification head.

Table 4.1: A list of arguments common to all scripts.

Argument	Values	Description
beta_2	float	An argument for the optimizer.
cle_dim	int	Dimension of character-level embeddings.
cont	0/1	Evaluating on the test data every epoch.
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 4.2: 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 for 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.
predict	string	Script returns only a prediction with 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 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 load and weights will be printed.

Table 4.3: 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 of result training data which should be formed by english IMDB dataset.
freeze	0,1	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 i-th fold will be used for test. It serves for running k-fold cross-validation in parallel runs.

Table 4.4: Arguments for `sentiment_analysis.py` script.

5. User documentation

6. Discussion

6.1 Experiments

This section offers a description of all performed experiments. For tagging and lemmatization exist two main experiment settings: with (morphodita) pipeline on top of bert embedding and simple version with bert plus simple classification head. Both networks can finetune (or can freeze bert layers.

jak trenovat (ktere vrstvy)? - netrenovat berta (jen embeddings) - vsechny - napred head a pak dotrenovat berta

Jaky zvolit learning rate? - stejny pro vse - jiny pro berta - schedulingy

jak vybrat vrstvy? - vybrat vse - 4 poslední - poslední - dynamicky (trénovat váhy) (-layers opšna att)

Velikost batche ktery berti model jaka data poskladat

Table 6.1:

lr 3e-05	1	2	3	4
lemmasRaw	98.60	98.59	98.58	98.58
lemmasDict	98.86	98.87	98.88	98.90
TagsRaw	97.52	97.42	97.54	97.50
TagsDict	97.71	97.65	97.77	97.73

Table 6.2:

lr 2e-05	1	2	3	4
lemmasRaw	98.63	98.62	98.63	98.61
lemmasDict	98.90	98.89	98.91	98.90
TagsRaw	97.59	97.59	97.68	97.68
TagsDict	97.79	97.79	97.85	97.88

Table 6.3:

lr 5e-05	1	2	3	4
lemmasRaw	98.53	98.49	98.48	98.47
lemmasDict	98.81	98.80	98.80	98.80
TagsRaw	97.21	97.14	97.19	97.14
TagsDict	97.46	97.39	97.45	97.40

Conclusion

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

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

[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. 1990. – Forschungsbericht

[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

[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

[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

[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. (2020), mar. – URL http://arxiv.org/abs/2003.10555

[Collobert et al. 2011] Collobert, Ronan ; Weston, Jason ; Bottou, Leon ; Karlen, Michael ; Kavukcuoglu, Koray ; Kuksa, Pavel: Natural Language Processing (almost) from Scratch. In: *J. Mach. Learn. Res.* 12 (2011), mar, S. 2493–2537. – URL http://arxiv.org/abs/1103.0398

[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`

[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`

[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. (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`

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

[Goodfellow et al. 2016] GOODFELLOW, Ian ; BENGIO, Yoshua ; COURVILLE, Aaron: *Deep Learning.* MIT Press, 2016. – `http://www.deeplearningbook.org`

[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

[Hana and Zeman 2005] HANA, Jiří ; ZEMAN, Daniel: Manual for Morphological Annotation Revision for the Prague Dependency Treebank 2.0 UFAL. 2005. – Forschungsbericht

[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

[Hercig et al. 2018] HERCIG, Tomáš ; KREJZL, Peter ; KRÁL, Pavel: Stance and sentiment in Czech. In: *Comput. y Sist.* 22 (2018), Nr. 3, S. 787–794. – URL http://nlp.kiv.zcu.cz/research/sentiment{#}stance.. – ISSN 20079737

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

[Horsmann and Zesch] HORSMANN, Tobias ; ZESCH, Torsten: Do LSTMs really work so well for PoS tagging?-A replication study. – Forschungsbericht. – 727–736 S

[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

[Huang et al. 2015] Huang, Zhiheng ; Xu, Wei ; Yu, Kai: Bidirectional LSTM-CRF Models for Sequence Tagging. (2015), aug. – URL `http://arxiv.org/abs/1508.01991`

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

[Hutchins] Hutchins, John: Two precursors of machine translation: Artsrouni and Trojanskij. – Forschungsbericht

[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. (2020), oct. – URL `http://arxiv.org/abs/2010.00454`

[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`

[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, Tomáš: Neural Networks for Sentiment Analysis in Czech. 2016. – Forschungsbericht

[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

[Li et al.] Li, Xin ; Bing, Lidong ; Zhang, Wenxuan ; Lam, Wai: Exploiting BERT for End-to-End Aspect-based Sentiment Analysis *. – Forschungsbericht. – 34–41 S

[Libovický et al.] Libovický, Jindřich ; Rosa, Rudolf ; Helcl, Jindřich ; Popel, Martin: Solving Three Czech NLP Tasks End-to-End with Neural Models. URL https://www.yelp.com/dataset/. – Forschungsbericht. – ISBN 11234/12839

[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. 2016. – Forschungsbericht

[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

[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.] McCulloch, WS ; Biophysics, W Pitts T. bulletin of mathematical ; 1943, Undefined: A logical calculus of the ideas immanent in nervous activity. URL https://link.springer.com/article/10.1007{%}252FBF02478259. – Forschungsbericht

[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=zzCzAMspYO{&}sig=xOHLubdLNN3Irw4cZUlmdyHK8BM

[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

[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 2009] Pan, Sinno J. ; Yang, Qiang: A Survey on Transfer Learning. (2009). – 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. – URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[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

[Plank et al.] Plank, Barbara ; Søgaard, Anders ; Goldberg, Yoav: Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss. URL `https://github.com/clab/cnn`. – Forschungsbericht

[Plisson et al.] Plisson, Joël ; Lavrac, Nada ; Mladenic, Dunja: A Rule based Approach to Word Lemmatization. – Forschungsbericht

[Putra et al.] Putra, Ilham F. ; Purwarianti, Ayu ; Ai-Vlb, U-Coe: Improving Indonesian Text Classification Using Multilingual Language Model. URL `https://www.yelp.com/dataset`. – Forschungsbericht

[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

[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. (2016), feb. – URL http://arxiv.org/abs/1602.03609

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

[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 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: Czech Text Processing with Contextual Embeddings: POS Tagging, Lemmatization, Parsing and NER. In: *International Conference on Text, Speech, and Dialogue* Springer (Veranst.), 2019, S. 137–150

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

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

[Sun et al.] Sun, Chi ; Qiu, Xipeng ; Xu, Yige ; Huang, Xuanjing: How to Fine-Tune BERT for Text Classification? URL `https://github.` – Forschungsbericht

[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

[pandas development team 2020] team, The pandas development: *pandas-dev/pandas: Pandas.* Februar 2020. – URL `https://doi.org/10.5281/zenodo.3509134`

[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

[Toutanova et al. 2003] Toutanova, Kristina ; Klein, Dan ; Manning, Christopher D. ; Singer, Yoram: Feature-rich part-of-speech tagging with a cyclic dependency network. In: *Proc. 2003 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - NAACL '03* Bd. 1. Morristown, NJ, USA : Association for Computational Linguistics (ACL), 2003, S. 173–180. – URL `http://portal.acm.org/citation.cfm?doid=1073445.1073478`

[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)

[Veselovská] Veselovská, Kateřina: *SENTIMENT ANALYSIS IN CZECH.* – 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. (2019), dec. – URL `http://arxiv.org/abs/1912.07076`

[Wang et al. 2015] Wang, Peilu ; Qian, Yao ; Soong, Frank K. ; He, Lei ; Zhao, Hai: Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Recurrent Neural Network. (2015), oct. – URL `http://arxiv.org/abs/1510.06168`

[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. (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. (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

[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

List of Figures

List of Tables

A. Attachments

A.1 First Attachment