



**FACULTY
OF MATHEMATICS
AND PHYSICS**
Charles University

MASTER THESIS

Bc. Petra Vysušilová

Czech NLP Processing 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 2020

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

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

In date

signature of the author

Dedication.

Title: Czech NLP Processing with Contextualized Embeddings

Author: Bc. Petra Vysušílová

Institute: Institute of Formal and Applied Linguistics

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

Abstract: Recently, several methods for unsupervised pre-training of contextualized word embeddings have been proposed, most importantly the BERT model (Devlin et al., 2018). Such contextualized representations have been extremely useful as additional features in many NLP tasks like morphosyntactic analysis, entity recognition or text classification.

Most of the evaluation have been carried out on English. However, several of the released models have been pre-trained on many languages including Czech, like multilingual BERT or XLM-RoBERTa (Conneau et al, 2019). Therefore, the goal of this thesis is to perform experiments quantifying improvements of employing pre-trained contextualized representation in Czech natural language processing.

Keywords: key words

Contents

Introduction	2
1 Theory	3
1.1 General Background	3
1.1.1 Label smoothing	3
1.1.2 Attention mechanism	4
1.1.3 Recurrent Neural Networks	4
1.1.4 Transformers	5
1.1.5 Embeddings	5
1.2 BERT explanation	6
1.2.1 Input embeddings	6
1.2.2 Pretraining tasks	7
1.2.3 Transformers encoder architecture	8
1.2.4 Why is BERT so special?	8
2 Implementation analysis	9
2.1 POS tagging and lemmatization model	9
2.1.1 Dataset and preprocessing	9
2.1.2 Network	10
2.1.3 Python implementation	11
3 Discussion	13
4 User documentation	14
Conclusion	15
Bibliography	16
List of Figures	17
List of Tables	18
A Attachments	19
A.1 First Attachment	19

Introduction

This work aims to improve some natural language processing (NLP) tasks for Czech with the use of recently published artificial intelligence (AI) state-of-the-art techniques. In Deep Learning book, NLP is defined as "...the use of human languages, such as English or French, by a computer." Goodfellow et al. (2016)[]. NLP offers a variety of problems to solve from oral-written language conversion, machine translation, syntax analysis used for automatic grammar correction as well as it serve as a base for further linguistic processing. Semantic analysis includes in addition to already mentioned machine translation other tasks like sentiment analysis, natural language text generation or recognition of homonymy or polysemy of given words. It could solve sophisticated assignments as answering questions about the input text document.

This work applies the most successful NLP methods of recent years to Czech NLP task - namely tagging, lemmatization and sentiment analysis. Tagging and lemmatization represent syntax analysis, in contrast with sentiment analysis which represents semantic type of task.

These tasks were chosen to show how pre-trained multilingual language models can help with different types of NLP tasks in one of languages model was trained on.

Theoretical background in NLP and used AI methods and related work is provided in the following chapter.

Implementation documentation in chapter 2 is followed by discussion about used methods and experiment results in chapter 3. Result models are accessible to user exploration as described in chapter 4 and text is closed by conclusion with future work proposals.

1. Theory

This chapter is divided into two parts. In the first part, general deep learning and natural language processing introduction is presented, with a focus of this work in mind. Second part of this chapter offers more detailed explanation of methods relevant to this work.

1.1 General Background

A history of neural networks started in 1940s (Goodfellow et al., 2016). Perceptron, found in 1957, is the simplest neural network with just one layer serving for binary classification. There also exist multi-class version for general classification. Problem of this perceptron was inability to classify data which are not linearly separable (see picture).

Deep neural networks era started in 2007 with bigger datasets and greater computational resources. These two new features opened possibility of neural network learning without expert handcrafted parameters tuning with good results. Some now frequently used ideas are quite old, like the perceptron or a back-propagation algorithm and are combined with better choice of activation functions, some ideas appeared in last two decades like convolutional neural networks or encoders. Basic type of DNN is multilayer perceptron (see picture). It is combined of neurons, every neuron has an activation function, which is applied to its input. Input to every, but first layer is weighted combination of (possibly selection of) previous layer neurons. Different types of NN differs by number and shape of layers, activation functions and connections between neurons. (see Some of key ideas for (not only) NLP and specifically BERT, are presented in the following subsections.

1.1.1 Label smoothing

~~Label smoothing is an idea applicable to every classification problem, therefore~~ is not limited to a NLP only. In vanilla 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. Instead of this, label smoothing applies following formula:

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

, where K is number of classes. That was for equation, but more important is the reason why to use it. Label smoothing is used as a cure for overfitting and overconfidence in the case of softmax as output activation function. Loss function for softmax classification is:

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

and after substitution of label smoothing¹:




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

, which gives after multiplication²:


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


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

1.1.2 Attention mechanism

Attention mechanism is widely used in NLP as a tool for extracting relevant informations from word sequences. For example when generating translation of some sentence, each word in target language corresponds to just few words in source sentence, not to a whole sentence. Attention gives weights to the words, which represents this connections. When the task is a question answering, attention can help a model to focus on relevant part in the text, where the answer is allocated. For example see picture  In Transformers architecture  which is relevant to this work, self-attention mechanism is used. This variant of attention do not connect one part of text (like question) to other different part of text (like answer), but only models relationships inside one sentence. Attention has three inputs – key, value and query, all vectors. The output is weighted sum of values and the weights are computed with respect to key-query pair. 

1.1.3 Recurrent Neural Networks

Language has a sequence nature. Word and sentences ordering is an important part of text and lack of it can make text absolutely nonsense. This places hard demands on NN, because  is different from most of other NN applications, where input samples are independent and ordering does not matter. Problem with vanishing/exploding gradient is even more fundamental, because text sequences can be very long and information from more distant words, which is relevant for given word, can vanish very easy.

The construction of such networks, which are able to capture natural language structure requires solving two problems  input should be understood by network as a sequence and there must be a possibility to use information from other parts of sentence (or text). First problem was solved by simple Recurrent Neural Networks (RNN) , while the latter problem needed more complicated approach. There are basically three attempts to solve this *short memory* of RNN cells – Gated Recurrent Unit (GRU), Long Short Term Memory (LSTM) and Transformers architecture with attention mechanism. Both LSTM and GRU uses an

¹Taken from: <https://leimao.github.io/blog/Label-Smoothing/>

²Taken from: <https://leimao.github.io/blog/Label-Smoothing/>

idea of gating. Gates work as weights for previous informations and new input. They help to decide which information should be remembered and which should be forgotten.

LSTM

LSTM cell uses input, output and forget gate. Every gate is composed by sigmoid function with actual input and previous hidden state as inputs.

GRU

GRU also uses gates: reset gate and update gate. Reset gate is responsible for how much of previous state will take in the new state. Update gate is then weight for a combination for previous and current states, which together form new output.

1.1.4 Transformers

A Transformers architecture was proposed in a paper **Attention Is All You Need** and essentially depends on **textbf{attention mechanism}** (see subsection 1.1.2). In the example of sentence processing, attention in nutshell is able to tell us which other words are important while determining something about currently processed word. Transformers uses encoder-decoder architecture, which was simultaneously proposed in 2014 by two papers. basic idea behind this architecture is following: encoder and decoder are connected by some middle layer of fixed size which aims to be good representation of an input. Encoder reads its input and tries to learn such weights, that encoder's final representation of the input contains all important information. Decoder has this representation as its input and tries to produce best results. It was first use for machine translation, so the output of the encoder in this case is a sentence in the target language with same semantics as original input. Transformers are novelty in the manner that there are no LSTM cells computing hidden states. **Picture** describes design of one layer. Both encoder and decoder are built by many blocks of respective type. Self-attention mechanism which is supposed to select most important words to be focused on is used on many places – for input of every encoder layer, input of every decoder layer (although masked) and also between encoder and decoder. On the decoder side, 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 representation of word position in the sentence (as it is not RNN cell and it can process all words simultaneously) Transformers uses position embeddings, which are trained to represent the ordering in the sentence.

1.1.5 Embeddings

Good performance of NLP models relies on text representation. What is hypothetically deserved is teaching computers to understand semantics of language. Once computer has a good representation of what given text *means*, it should be easy to answer questions, translate into another language etc. Because NN

are able to work only with numerical representation of inputs, second requirement upon such language representation is to be numerical. Straightforward way is to represent input words in one-hot encoding. It means that single word is represented by vector where is only 1 at the position of respective word and all other positions are 0s. Such vector is long as a number of distinct words in a dictionary, so it could be quite wide. Most important problem is that in this representation, every two words has same distance to each other. This is not true in linguistics. Words can have similar meanings, be opposite to each other etc.

Better solution than one-hot encoding are embeddings. Embedding is a fixed-size vector, which represent input word. These embedding are not hand crafted by humans, they are learned by NN. They can be learned for every specific task from scratch or it is possible to use good pretrained embeddings. Before contextualized embeddings appeared, such pretrained embeddings were created mainly by Word2Vec, specifically by its two variants: CBOW and SkipGram model. CBOW model's objective is to predict a word from its context and SkipGram does exactly the opposite - predicting context of a given word. These predicting objectives serves just as a tool for forcing a network to learn useful word representation. There is an embedding layer at the beginning of the network with size $number_of_words \times embedding_size$. Word2Vec-like embeddings uses just few context words and they does not use explicitly information about statistics in whole dataset. GloVe 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 though uses context of the word – it is in fact how they are meant to work. Similar words are supposed to appear frequently in similar context. Problem of such embedding is that a word has always same embedding independently of a context. This is definitely a problem in the case of homonyms, but contextualized embeddings showed to be generally better than traditional embeddings. That is why contextualized embeddings were invented in recent years. In addition to BERT, there is also GPT-2 and ELMo. Their comparison can be found in a subsection 1.2.4.

As embeddings are trained after the input is encoded into one-hot vectors, it is impossible to use pretrained embedding for encoding of previously unseen words. This is solved by embeddings of characters or subwords.

1.2 BERT explanation

This work mainly relies on Bidirectional Encoder Representations from Transformers (BERT), so this section describes and explains main mechanisms of this model. The core of BERT algorithm is based on these three features – two unsupervised task for pretraining, input and its embeddings, transformers encoder architecture.

1.2.1 Input embeddings

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

Token embeddings

Input of BERT model can be one or two word sequences (not necessarily two sentences, but e.g. also paragraphs). All words are split into tokens and converted into embeddings with a use of pretrained embeddings model. One word can be tokenized into more tokens as WordPieces embeddings are used. WordPiece pre-trained embedding algorithm was originally created for task of Google voice search for Asian languages and is designed to minimize occurrence of unknown words tokens. This model was not pretrained as a part of BERT paper experiments, but represents quite interesting solution, so I will briefly explain the idea. In the first iteration of training, the model creates embeddings only for basic characters. In every other iteration, some existing model words are concatenated together in the way that cause 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 on figure.

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

Position embeddings

All input tokens are processed simultaneously, so there is no information about order of tokens. However, nature of the language is sequential, bunch of words without an order has no language meaning, so there are position embeddings for this. They have same shape as token embeddings (and also as segment embeddings), so concatenation with two other types is easy.

Segment embeddings

These embeddings just indicates whether the token belongs to first or second part of input. has same dimension as position and token embeddings.

1.2.2 Pretraining tasks

BERT is pretrained on two unsupervised tasks – Next Sentence Prediction (NSP) and Masked Language Modeling (MLM). These two tasks were chosen specifically because BERT's authors believe they can force language model to learn general and useful knowledge about language.

Next Sentence Prediction

Input of the BERT model for this task are two sentences A and B. In 50% of cases, sentence B is sentence which really follows sentence A in the text. Otherwise it is random sentence from the text. A goal of the task is decision whether sentence B is following or random i.e. binary classification. Motivation for this task is a need to represent relationships between sentences, not only between words. Experiments showed its usefulness for text tasks as question answering. Last

hidden representation of CLS token, therefore its representation in last hidden layer, is used for prediction.

Masked Language Modeling

Masked Language Modeling or Cloze tasks consist of prediction of some missing words in text. In the case of BERT, its implementing in the following way: 15% of tokens in each sequence are chosen. For each of this chosen tokens there is 80% chance to be replaced by MASK token, 10% chance to be replaced by random token or it will remain unchanged with 10% probability. This ensures, that the model will try to predict tokens not only in the case of MASK token presence. For backpropagation, only predictions of MASK token are taken into account. Prediction is done by softmax which input is last hidden representation of respective token. This prediction is done with informations about both sides context, which is main difference from other preceding language models. How exactly is this allowed will be described in the architecture section.

1.2.3 Transformers encoder architecture

BERT adapts encoder part of architecture from original Transformers paper. Actually, BERT uses this encoder architecture for each layer, and there is L (e.g. 12 in the case of BERT base model) encoder layers finished with one fully connected layer for specific task. In pretraining, there is only one layer responsible for MLM and NSP, as each task uses different part of last hidden layer informations.

1.2.4 Why is BERT so special?

For deep learning architectures, it is quite common that reason why they work (especially why they work so well in comparison with other possibilities) is not visible at first sight. Authors offers following explanation. In contrast to other SOTA league language models, BERT uses both bidirectional context of words. In the figure is a comparison of two other such models - ELMo and GPT. GPT is also based on Transformers architecture, but unlike BERT, it uses decoder part of Transformer. GPT uses just left context of the word, as a sentence is proceeded sequentially. ELMo uses LSTM with both left-to-right and right-to-left training and concatenates results at the end. BERT is proceeding both sides context simultaneously and due to an architecture, in every layer every transformer block has potentially information from every other block (and thus from every other word). That is the reason, why BERT is called **deeply** bidirectional or also non-directional (because there is no right-to-left or left-to-right direction of processing).

Explanation of why and how BERT works are, however, not so simple. There are many papers on how BERT works, what BERT really learns and know, how BERT architecture can be simplified and so on. There actually appeared new term "bertology" for this area of research.

2. Implementation analysis

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.

~~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., 2019) 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.~~

doplnit
diskuzi
o
různých
možnostech
definice

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., 2019). 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 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.

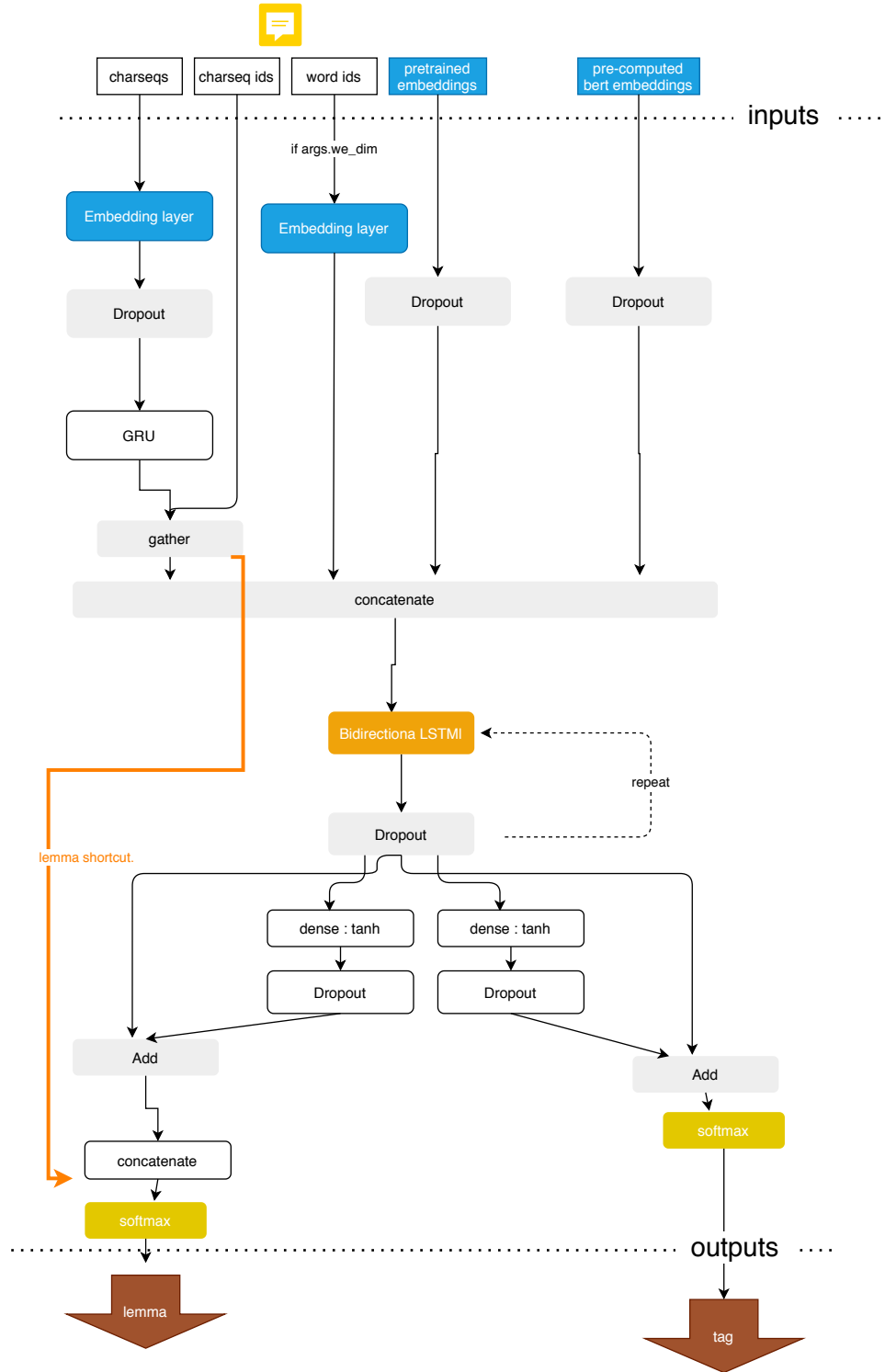
2.1.2 Network

Lemmatization as well as tagging are considered to be classification tasks (same as in (Straka et al., 2019)), i.e. lemmatization classification target are lemma generating rules and tagging target is tag, ~~or lemma~~. 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.

2.1.3 Python implementation

Model is implemented in Python (specifically python version 3.6.9). All dependencies and used libraries are listed in the requirements.txt file, but I should specifically mention library Transformers , which contains pretrained bert models and tools for their usage as tokenizer. (In this model, I used

uncased version of multilingual models .
Code for all models is available as an attachment of the work.

3. Discussion

New York (NY)

4. User documentation

Conclusion

Bibliography

- [Goodfellow et al. 2016] GOODFELLOW, Ian ; BENGIO, Yoshua ; COURVILLE, Aaron: *Deep Learning*. MIT Press, 2016. – <http://www.deeplearningbook.org>
- [Straka et al. 2019] 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

List of Figures

List of Tables

A. Attachments

A.1 First Attachment