FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Word sense representation for the Czech language
Student:	Vojtěch Paukner
Supervisor:	Ing. Karel Klouda, Ph.D.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2019/20

Instructions

Recent progress in natural language processing (NLP) allows attacking more sophisticated linguistic tasks. One of them is the detection of distinct word senses. The goal of this work is to apply recently introduced methods on the Czech language that is, as a fusional language, usually harder to approach.
1) Get familiar with the MorphoDiTa tool for preprocessing Czech textual data and with NLP for the Czech language in general.
2) Survey state of the art methods for word sense representation, including.
3) Apply selected methods (including those in references [1] and [2] below) on a Czech corpus and evaluate results.

References

[1] Peters ME, Neumann M, Iyyer M, Gardner M, Clark C, Lee K, Zettlemoyer L. Deep contextualized word representations. arXiv preprint arXiv:1802.05365. 2018 Feb 15.
[2] Melamud, Oren, Jacob Goldberger, and Ido Dagan. "context2vec: Learning generic context embedding with bidirectional lstm." Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning. 2016.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 5, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Word sense representation for the Czech language

Vojtěch Paukner

Department of Applied Mathematics
Supervisor: Ing. Karel Klouda, Ph.D.

May 15, 2019

Acknowledgements

I wish to express my sincere gratitude to Ing. Karel Klouda, Ph.D., for valuable guidance and sharing expertise and for providing me with the additional language datasets. These have allowed me to extend the scope of my thesis and to take the full advantage of the state-of-the-art methods on the larger dataset. I would also like to thank my family and friends for supporting me during the creation of this thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 15, 2019 ……

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Paukner, Vojtěch. *Word sense representation for the Czech language.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Tato práce detailně zkoumá tradiční a moderní metody zpracování přirozeného jazyka. Zvláštní důraz je kladen na jazyky s rozmanitou morfologií. Nejmodernější metody jsou pak aplikovány různými způsoby na český jazyk s cílem rozlišit jednotlivé významy slov na základě příkladů jejich užití ve větách. Důležitou součástí práce je vyhodnocení těchto experimentů.

Klíčová slova zpracování přirozeného jazyka, strojové učení, český jazyk, rozlišení jednotlivých významů slova, neuronová síť

Abstract

The thesis surveys traditional and state-of-the-art methods of natural language processing. Particular importance is placed on languages with rich morphology. The state-of-the-art methods are then applied in various ways on the Czech language in order to differentiate between distinct word senses based on their usage in a sentence. Evaluation of these experiments is an important part of the thesis.

Keywords natural language processing, machine learning, Czech, word sense disambiguation, neural network

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Natural language as the primary form of interaction between humans is crucial to be understood by computers in order to achieve human-like software user experience. The vision of getting rid of conventional input devices such as mouse or keyboard as well as the current user experience which is provided almost entirely by a graphical interface has been pushing natural language processing technologies fast forward in the recent years. The general idea has been to replace the current computer interface with a more natural way of intuitive voice communication between the user and the software.

Accurate voice recognition algorithms have already become a standard and a critical feature of artificial voice assistants which have been making their way to phones, houses or automobiles. However, merely transcribing voice instructions to text ones is far from knowing what the user wants. As a consequence, many machine learning algorithms from the field of natural language processing focused on understanding the meaning of sentences or even long conversations have been introduced lately.

In natural language, a word can have multiple meanings. Disambiguation between different meanings of a word is a challenging problem not only for computers but in some cases for humans as well. Machine learning algorithms capable of differentiating between distinct word meanings (so-called word disambiguation task) have been proposed just recently. It must be seen that this task is very hard to be attacked by computers and even the state-of-the-art machine learning algorithms are far from being perfect.

The word sense disambiguation task can be approached in supervised and unsupervised manners. Unfortunately, context differences between distinct word senses sometimes tend to be very small making it hard to use unsupervised algorithms. To use supervised algorithms on the word sense

disambiguation task, a vast amount of usage examples of a word in a sentence must be manually created by humans. Moreover, while creating such dataset, it can be challenging to decide on the level of word sense separation (in some dictionaries "storm" in colloquial language and "storm" in scientific writings are of different meanings – which is probably not what a typical human would say).

The goal of this thesis is to survey the state-of-the-art methods attacking the word sense disambiguation task and natural language processing in general, try to apply them on Czech and German language and evaluate the results. I will first introduce the major challenges in natural language processing to the reader and explain linguistic characteristics of fusional languages (such as the Czech language). Afterwards, I will survey the traditional and state-of-the-art machine learning algorithms used in the field of NLP. Finally, I will apply the introduced methods to Czech a German language and evaluate the results.

Chapter 2

Natural language processing (NLP)

Natural language is often hard to be correctly understood by computers. Although it is, in fact, a symbolic signalling system [1], it may use prefixes as well as suffixes to generate situation-specific versions of words or it might even use character-level word deviations to express further subjective feelings[1].

Moreover, many times full information or some parts of it are not expressed in the sentence at all, and it is up to the reader to figure them out based on the context[2]. Often a certain language symbol (word) can be assigned to multiple word meanings[3] which is also making it harder for computers to understand the sentence since a simple dictionary of words and their meanings is not sufficient.

In this chapter, the basic concepts from linguistics which are important to be understood for later usage in machine learning algorithms will be introduced. Furthermore, the traditional, as well as the modern statistical approach to language modelling, will be presented.

2.1 Levels of natural language comprehension

As it has been already mentioned, natural language is hard to be approached mostly due to its ambiguousness and its relation to the particular situation, medium or even the speaker's characteristic. To build a computer system able

[1] For example if something is "huuuuuge" it probably means it is bigger than just "huge".

[2] Typical example is when the subject of a sentence has been left out since it has been mentioned in some of the previous sentences. Sometimes, it is not expressed at all (i.e., It is raining.).

[3] Symbol "bank" can be the financial institution as well as the area by a river.

to understand human language sufficiently, the input data (text) needs to be analysed on multiple levels, each dealing with a different linguistic analysis task. The following categorisation has been inspired by [1].

2.1.1 Morphological analysis of natural language

Sentences are composed of basic units – words. Boundaries between distinct words in natural language are sometimes hard to be defined – for example in a phonetically represented sentence, defining these boundaries can be challenging since words might directly follow each other. In some languages, words can be distinguished by their vowel's harmony (i.e., Turkish). More often, a morphological attitude (which can also be applied to written textual data) is used. If the symbol (substring from the analysed string - spaces do not matter) in a sentence comply with the three following rules, it is considered to be a stand-alone word: positional mobility (symbol can be moved to a different position of the sentence), uninterruptability (no other extraneous information can be positioned between individual morphemes of the symbol), internal stability: there is a fixed order of morphemes inside of the symbol's structure. [2]

However, in most languages, words can also be compositions of other words, suffixes or prefixes - so-called morphemes[4]. From this point of view, natural languages are divided into two categories: analytical (isolating) and synthetic ones. [2]. Being able to correctly analyze individual morphemes of a word is important to understand the word itself and also its context.

2.1.1.1 Analytical languages

Analytical (isolating) languages are the simplest ones as the composition of words goes. Words are made of one or more free morphemes[5] and no bound morphemes[6]. No morphemes are expressing, for example, the tense (i.e., future) or the grammatical number (i.e., plural). [4] The sentence "John has not bought the apples" would be expressed as "John not buy apple" in an analytical language. Mandarin Chinese is an example of such a language.

2.1.1.2 Synthetic languages

Synthetic languages construct words consisting of free as well as bound morphemes. Bound morphemes (prefixes, suffixes) add information about the

[4] "Morpheme is the smallest unit of language that conveys some meaning."[3]

[5] Free morphemes are ones which can be used as stand-alone words. For example "unhappiness" is made of three morphemes: "un", "happy" and "ness". The morpheme "happy" is the free morpheme.

[6] Bound morphemes are opposites of free morphemes. Bound morphemes are not stand-alone words on their own. For example, in the word "unhappiness" morphemes "un" and "ness" are bound morphemes.

tense, grammatical number or for example the gender to the word. Synthetic languages are categorised in three groups based on the level of variety and complexity of their morphemes: agglutinative, fusional and polysynthetic languages. [2]

Words in agglutinative languages may consist of one or more morphemes. However, boundaries between distinct morphemes are easy to be found. Each morpheme represents single meaning, and they do not fall loosely together. Example of such language is Turkish. [2]

Fusional languages have words constructed of one or more morphemes. However, unlike agglutinative languages, their morphemes can express multiple pieces of grammatical information each. [2] Moreover, distinct morphemes might fall loosely together or end up in a new morpheme[7]. As a consequence, distinguishing between individual morphemes can be a complicated task. Slavic languages are typical examples from this category.

Polysynthetic languages usually tend to build long words containing many morphemes of distinct meanings. They also use many prefixes and postfixes and might contain multiple stems[8] in a single word. West Greenlandic is an example of such a language. [2]

2.1.2 Syntactic analysis of natural language

Having already analysed character-level deviations and morphological constructions of words, the focus can move to the word level analysis. The syntactic analysis examines the order of words in sentences, their relations (structure) and syntactic meaning (parts of speech). Dependency trees are usually used as a visualisation of syntactic analysis results. There exist multiple syntactic dependency tree datasets for supervised NLP tasks such as the famous Penn Treebank dataset [5].

2.1.3 Semantic analysis of natural language

The goal of the semantic analysis is to use results from previous layers and understand the true meaning of phrases or even whole articles. Since natural languages are ambiguous and widely context dependent, semantic analysis algorithms are still in the early stages of development, and their results are still far from perfect.

Moreover, not all the information is always kept in the input sentence. Sometimes it might be necessary for the algorithm to be familiar with the

[7] For example in the Czech language concatenation of morphemes “ne” and “je” results in “není”.

[8] Part of a word which is common to all its inflexions.

real world's facts to understand its input. In this work, algorithms designed for semantic analysis will be introduced.

2.1.4 Further discourse processing

As it has been already pointed out, further information which is needed to understand a particular sentence or an article might be stored somewhere else than the input text itself. This information might be some general knowledge data (i.e., water is drinkable), cultural awareness (i.e., The Good Soldier Švejk has been written by Jaroslav Hašek), the speaker's background, emotions or even the situation where the sentence/article has been used. It is uncertain if this information can even be completely described by computers. As a result, this level of language understanding is here mentioned as a future outlook.

2.2 The evolution of natural language processing

In the early stages of natural language processing, the first and foremost attitude used to understand the structure of the language was utilising rule-based systems. Such attitude was often using context-free grammars or regular expressions to model constituency[9] between words in sentences [6]. Rules of these systems were manually created by linguists and to build a reasonably robust solution a massive number of different rules had to be created. This also made the systems hardly scalable and their sets of rules (which were accumulating in multiple levels on the top of other) chaotic. As a consequence, such development was getting exceedingly expensive and also the results were not that satisfying as the systems could not gracefully transfer to a different textual domain[10].

As a result, the statistical approach (language modelling[11]) to language analysis has been proposed. This "observational" attitude has not needed that many human resources since instead of applying human-designed rules it has learned to analyse the language from a vast amount of textual data, which are easy to be acquired (i.e., web sites, books), itself. However, today's solutions usually combine the traditional rule-based system approach with the statistical one to achieve state-of-the-art accuracy.

[9] Grouping words which go together.

[10] For example, rule-based systems built on academic-style language were not suitable for colloquial ones.

[11] Language models learn a probability distribution over a sequence of symbols/words of a particular language.

2.2.1 Statistical language model

Traditional statistical language models use localist representation [12] of a word based on a discrete dictionary. Their general goal is to assign a probability of the occurrence of a particular word sequence:

$$P(w_1, w_2, w_3, w_4) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2)P(w_4|w_1w_2w_3),$$

where w_1, w_2, w_3, w_4 are words. [7] To get the probability of the presence of a certain word sequence A conditioned by previous context sequence B a simple frequency attitude can be used:

$$P(A|B) = \frac{count(A) + count(B)}{count(B)}.$$

However, full estimation based on the whole sentence or even an article (for a broader context) can be computationally very expensive. Moreover, such an attitude shows its downsides when the word counts in the text are rather low[13]. [8]

One of the possible solutions to the full estimation problems is called the n-gram model which is based on the Markov chain stochastic model. [9] It generally assumes that to determine the probability of a word occurrence based on its context, only the previous $n - i$ words need to be taken into account.

The simplest example of the n-gram model is the uni-gram model ($n = 1$) which takes into account only the current word and does not look at the preceding sequence at all. That makes it just counting the probabilities of each word's occurrence

$$P(w_i|w_1...w_{i-1}) \sim P(w_i) = \frac{count(w_i)}{\sum_{\tilde{w}} count(\tilde{w})},$$

where $w_1, w_2, ..., w_i$ are words and $\tilde{w}$ is an array of all the words except the word w_i. [8]

However, in practice n-gram models of $n > 1$ are used:

$$P(w_i|w_1...w_{i-1}) \sim P(w_i|w_{i-1}...w_{i-n}).$$

[12] Each word is represented by its position in the dictionary. Localist representation is usually implemented by a vector which contains all zeros except a one at the position of the particular word in the dictionary (one-hot vector encoding).

[13] For example, having the only training dataset sentences "John was born in Prague." and "John got lost in Brno" results in a zero probability of the presence of the word "Brno" conditioned by context "John was born in".

Such n-gram models are usable for simple language analysis tasks mostly on analytical (or close to analytical) languages or textual data which has been preprocessed by some stemmer or lemmatiser[14].

Another problem with the traditional statistical approach to language processing is related to the words which have not been included in the training dataset and yet appear in later input textual data. Therefore their probability would always be zero. A possible solution is to add one to the occurrence count of each word (Laplace-smoothed n-grams) making it possible to take into account new words. [10]

2.2.2 Neural language model

The above-explained traditional statistical language model has at least one serious issue: because of the localist word representation, it lacks the notion of the word. Therefore, there is no information about the relationships between the words encoded in their representations.

However, to correctly analyse natural language, the model should be able to recognise at least some similarity between words (i.e., a car and an automobile are closer together than a car and a cat). There could be two possible solutions – one would be using a human-made dictionary containing synonyms and homonyms, but as it has been already discussed in this work, such attitude is not easy to implement and to maintain.

As a consequence, a representation of a word which already encapsulates the notion of the word has been proposed. Such word vectors are dense, their dimension is usually a hyper-parameter of the model, and they allow to inherently measure distances between particular words.

These dense vectors (called *word embeddings*) distribute the meaning of the word over the whole space of word embeddings. As a consequence, each dense vector must be good at predicting other words' vectors in its context. Moreover, changing the context means the particular word embedding changes as well. [11]

Dense word embeddings are learned by neural networks. There are multiple ways to learn these embeddings, but generally, two attitudes take place. One is based on predicting the centre word according to the known context. The other one does the exact opposite – it predicts the word's context. Both the attitudes are unsupervised, so they take advantage of the vast amount of textual data available.

[14]Lemma of a word is its dictionary form.

2.3 The Czech language processing

Czech as a flexible fusional language is challenging to be approached by computers. There are multiple properties of the language which make it hard. First of all, Czech language contains seven grammatical case inflexions which indicate the use of a particular noun in a sentence. Moreover, since the grammatical function of the word is already encoded in its form, the language in its colloquial style can use the words in multiple orders and still maintain the same meaning[15]. Affixes (word prefixes and suffixes) in the Czech language contain information about the word's context, and they are not usually easy to be separated since they tend to fall loosely together.

Nevertheless, Czech is a pronoun dropping language which means it can leave out a pronoun if it is obvious from the context[16]. These and other more detailed deviations make the Czech language hard to be understood by computers. [12]

A few institutions are trying to bring the Czech language digital dictionaries, corpora, traditional language processing tools as well as natural language machine learning based tools to the public. For example, the Institute of the Czech Language[17] has already introduced multiple on-line Czech explanatory dictionaries, corpora and many specific smaller dictionaries – such as the dictionary of Czech affixes. However, these dictionaries are usually only a digital copy of the printed version which means they are not well-suitable for automated processing by a computer. Another institution which significantly contributes to the research of the Czech language is the Natural Language Processing Centre at the Faculty of Informatics, Masaryk University, Brno[18]. This institution has released multiple computer processable lexical databases (e.g., Czech WordNet, Czech Lexical Database) as well as multiple tools for Czech language parsing and syntactic and semantic analysis. They are also trying to take usage of machine learning applied to the Czech language. The Czech National Corpus [13] published by Faculty of Arts at Charles University in Prague is one the biggest Czech corpora containing more than 10 million words and including parallel corpora to with translations from or to 30+ languages.

[15] Sentences "Martin vypil nápoj.", "Nápoj vypil Martin." and "Martin nápoj vypil. " mean the same (Martin has drunk a beverage.) although the order of words is different.

[16] i.e., "Martin vypil nápoj. Už nemá žízeň." The first sentence says "Martin has drunk the beverage", but the second sentence completely leaves out the Martin's or any pronoun describing Martin, although it is related to Martin ("He is not thirsty anymore.").

[17] `http://www.ujc.cas.cz`

[18] `https://nlp.fi.muni.cz/en/NLPCentre`

Finally, the Institute of Formal and Applied Linguistics[19] whose tool and corpus will be used later in this work, has already introduced a vast amount of tools for syntactic and semantic analysis, machine translation, lexicons or text processing. It also regularly publishes updated Czech language corpora.

2.3.1 MorphoDiTa tool

MorphoDiTa is a state-of-the-art open-source natural language processing tool focusing merely on languages with very rich morphology introduced by the Institute of Formal and Applied Linguistics at Faculty of Mathematics and Physics, Charles University in Prague [14]. The following description of this tool is based on the paper [14] by which the tool has been presented. The tool performs lemmatization[20], morphological analysis[21], morphological generation[22], tokenization[23] and tagging[24].

Under the hood of the tool is an extraordinarily efficient combination of existing NLP algorithms. Since it focuses mainly on fusional languages, it uses an efficient way to deal with a large number of word endings per each lemma by introducing so-called "morphological templates". These templates are created in unsupervised manners without any linguistic knowledge about the particular language. They use special tree structures ("tries" [15]) where each node corresponds to a character and descendants of each node share the same prefix to find common templates as the suffixes of words go.

The main idea is to split the words into individual characters, build a "trie" of them and then find sub-trees which are as deep as possible (the longer the prefix is the better since the algorithm is looking for a prefixes which will be common to most word forms possible) as long as their sub-trees' lengths are smaller than N (which is a hyper-parameter of the model, the suggestion is that suffixes are usually short). An example of such a tree is visualised in Figure 2.1.

Representation of different word inflexions by the templates explained in the previous paragraph makes the algorithm very efficient in terms of computational complexity as well as computer memory requirements. All the possible lemmas from the "tries" are then passed to the next layer

[19] `https://ufal.mff.cuni.cz/`

[20] Generating a sequence of word lemmas based on the input sentence. For example the Czech sentence "Staročeské knihy jsou čtivé." would result in a sequence of lemmas ["staročeský", "kniha", "být", "čtivý"].

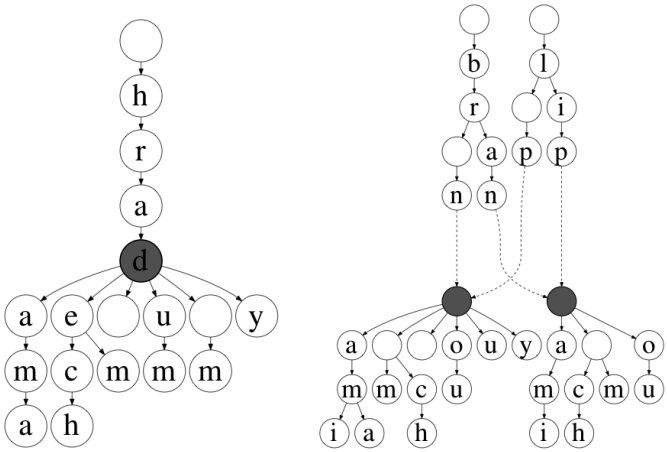
[21] Analysis of the way the word was created from individual morphemes.

[22] A tool which generates a word form based on a lemma and description of the demanded form.

[23] Splitting sentences into an array of word and symbols.

[24] By tagging individual words of the sentence, additional information such as the part-of-speech, word form or other contextual information are added to each word.

Figure 2.1: So-called "tries" which are built inside of the MorphoDiTa tool. On the left, there is a character level decomposition of the Czech word "hrad" (castle) based on its different forms which occurred in the training corpus. On the right two lemmas sharing the same way of creating inflexions ("template"). Figure taken from [14].



of MorphoDiTa which is a supervised POS[25] tagger implemented as an averaged perceptron [16]. Lastly, the supervised named entity recognizer[26] based on [14] is applied.

MorphoDiTa comes with robust pre-trained language models. However, users can train their own models on any language. The tool is implemented in C++ and offers bindings to Python, Java and Perl. For users who do not want to run it locally, an online version, as well as REST service API, are available. The tool itself is computationally very efficient and can handle to process a vast amount of data in a short period.

[25] Part-of-speech.

[26] Analysis of entities in a sentence. For example, in the sentence "Martin bought products from Apple yesterday." the analysis would find out that "Martin" is a person, "Apple" is a company and "yesterday" is the time of the action.

Chapter 3

Theoretical background

In this chapter, the essential machine learning algorithms that need to be known for a later explanation of the state-of-the-art methods for distinguishing word senses will be presented.

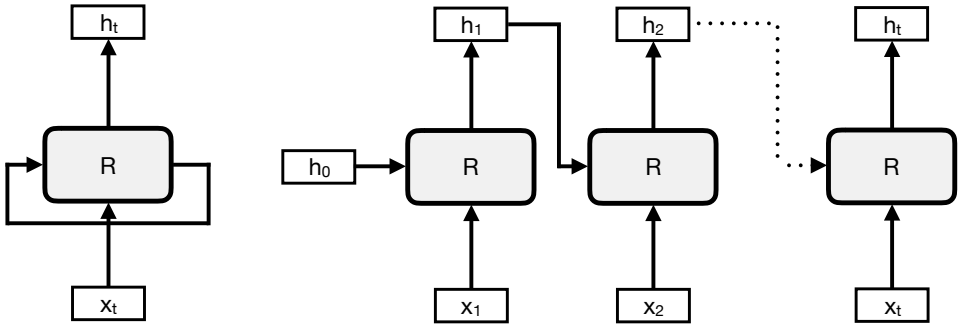
3.1 Recurrent neural network (RNN)

In the natural language, a word can have multiple meanings depending on its context and use in a sentence. To be able to recognise the sense of the particular word in a sentence the whole sentence has to be taken into account. A sentence can be seen as a sequence of words and the general idea behind understanding the word's context is to create word vectors which encapsulate words occurring before as well as after the particular words in the right order.

To learn a word representation while taking into account also a sequence of preceding and following words a traditional feedforward neural network cannot be used. For each input into a basic feedforward neural network, an output completely independent on previous inputs is generated. For this reason, basic feedforward neural networks are not suitable for sequence modelling. As a consequence, recurrent neural networks [17] has been proposed addressing this problem.

The objective of the recurrent cell in the recurrent neural network (see Figure 3.1) is to generate an output based on the input at the current time step of a sequence influenced by the inputs from previous time steps. In the recurrent neural network, this is achieved by maintaining a hidden state vector which "summarises" previously processed inputs. The recurrent cell passes the hidden state vector back to itself together with the input at a given time step of the sequence. Therefore, the recurrent cell takes two inputs at each time step t: input vector at time step t and the

Figure 3.1: Compact visualisation of the recurrent neural network is on the left. Over time unrolled visualisation for a better understanding of the flow of the recurrent neural network is on the right. R is a recurrent cell with weight matrix $\mathbf{W}$, h_1,h_2,...h_t are hidden state vectors at each time step t, h_0 is the hidden state initialisation vector and x_1,x_2,...x_t are input vectors at each time step t. Image created by the author of this thesis.



hidden state from the previous time step $t-1$. The output of this cell is the hidden state at time step t which can be further transformed depending on a particular task or nothing but passed again to the recurrent cell along with next time step input.

Let s_h be a fixed size of the hidden state vector (user-defined) and s_d a fixed dimension of the input encoding. Recurrent cell is a single perceptron. Inside of the recurrent cell at each time step t a sum of the input vector and the previous hidden state vector both weighted by corresponding weighting matrix is passed into a non-linear hyperbolic tangent function. Mathematical expression of the new hidden state computation:

$$h_t = \tanh(\mathbf{W}_h h_{t-1} + \mathbf{W}_x x_t + b),$$

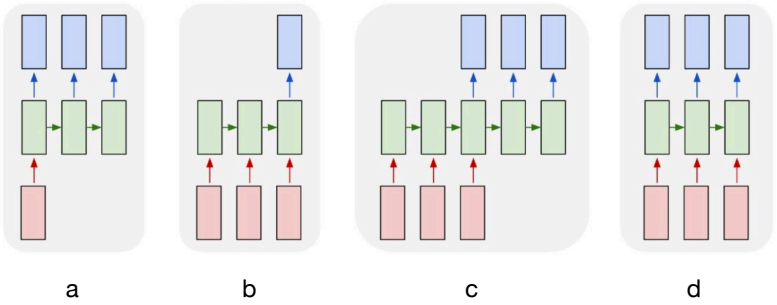
where $h_{t-1} \in \mathbb{R}^{s_h}$ is the hidden state vector at time step t, $\mathbf{W}_h \in \mathbb{R}^{s_h,s_h}$ and $\mathbf{W}_x \in \mathbb{R}^{s_h,s_d}$ are learned weight matrices and $x_t \in \mathbb{R}^{s_d}$ is the input vector at time step t and $b \in \mathbb{R}^{s_h}$ is a bias. Weight matrices contain the same values for all the time steps. If the hidden state at time step t is also used as the output of the network, further transformation can be applied

$$y_t = \mathbf{W}_y h_t,$$

where $\mathbf{W}_y \in \mathbb{R}^{s_h,s_h}$ is the learned weighting matrix. This transformation can be used for example for classification task (predicting class based on hidden state vector continuous values).

Recurrent neural networks can be used in multiple ways (Figure 3.2) based on the way of handling their input and output. "One to many" model takes

Figure 3.2: Different usages of the recurrent neural network. a: one to many (e.g., generating captions of a picture), b: many to one (e.g., sentiment analysis), c: many to many after receiving complete input sentence (e.g., machine translation), d: many to many in "real time" (e.g., video frames captioning) Visualisation borrowed from [18].



the input only at the first time step and generates an output sequence of the desired size. This is often used for the image captioning task[27].

On the other hand, "many to one" model generates just one output based on a sequence of inputs. Sentiment analysis[28] is a typical use of this model since the hidden state vector of the last time step encapsulates to some extent the previously received inputs.

"Many to many" models both receive the inputs at each time step and both use the hidden state vector at each time step to generate their output. One model generates an output at each time step of the input sequence – this can be used for video classification based on individual frames. The other one waits for generating the output sequence until the entire input sequence is passed. This makes the model capable of understanding the input sequence as

[27]Image captioning task is about describing visual content (images) by a sequence of words which can be more easily processed by search engines. For example given an image of a cat sitting on a desk the objective of the image captioning model is to predict a sequence of words "cat sitting on a desk". Google achieved excellent results in this task using neural networks in [19].

[28]Sentiment analysis is a task from the NLP field which tries to extract subjective information from the text. It can be used for example to gain data about customers' satisfaction with the product by analysing their comments regarding the product on social media.

a whole. The output sequence of a variable length is then generated. Machine translation[29] is a typical use of this architecture.

3.1.1 Long short-term memory network (LSTM)

Unfortunately, traditional recurrent neural networks tend to "forget" information over a long period. Although, in theory, RNNs can learn long term dependencies, in practice they almost cannot. This issue has been addressed mostly to problems with vanishing gradients during the training process [21].

The idea behind the solution to the problem of RNNs forgetting long term dependencies is inspired by the way a human brain processes information. Humans do not "store" everything they perceive directly in their brains [22]; on the contrary, they pick out the critical information out of it and store just this. A similar approach to information processing in recurrent neural networks has been proposed in [23] and the resulting neural network architecture has been named Long short-term memory network (LSTM).

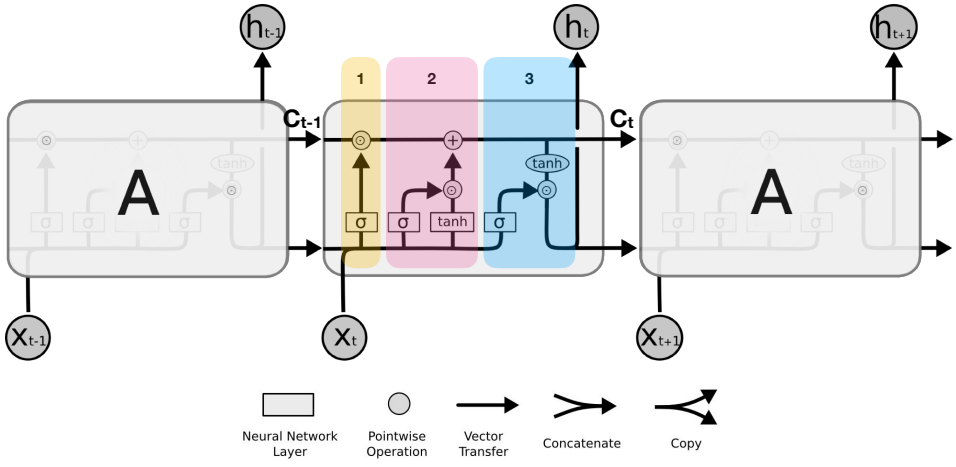
Long short-term memory network presents a new recurrent neural network cell (Figure 3.3) with so-called gates enabling the cell to decide which new information should it learn and which of already learned information should be forgotten. Whereas in traditional simple RNN cell one perceptron is found, in LSTM cell there are four of them formed in three "gate layers". In addition to the RNN cell, the LSTM cell maintains its inner cell state at time step t as the vector c_t. Crucial information is stored in this vector as well as the no longer important one is removed from it. Other than that, the hidden state vector, input vectors and the overall flow of the basic RNN network remains the same with LSTM.

In the first, so-called "forget gate layer" the LSTM cell decides which information of its cell state vector c_t should be forgotten. This is done by running previous hidden state vector h_{t-1} and current input vector x_t through a perceptron with a sigmoid activation function. The "forget gate layer" output vector is f_t:

$$f_t = \sigma(\mathbf{W}_{fh} h_{t-1} + \mathbf{W}_{fx} x_t + b_f),$$

[29] Machine translation (MT) algorithms try to understand multiple languages and perform translations between them without the help of a human. Traditional MT algorithms were based on training data consisting of the same text written in different languages. However, current state-of-the-art methods [20] learn from independent text training data per each language and try to understand them. This leads to creating "inter-language" representations of sentences using encoder-decoder architectures. Unfortunately, it still has not been shown if a universal language representation (applicable for every natural language) can be created.

Figure 3.3: Visualisation of the internal implementation of the recurrent LSTM cell. Individual "gate layers" are coloured differently. Yellow rectangle highlights "forget gate layer". Pink rectangle displays "input gate layer". Blue rectangle shows "output gate layer". Original visualisation borrowed from [24], edited for this work by the author of this thesis.



where $\mathbf{W}_{fh} \in \mathbb{R}^{s_h,s_h}$ and $\mathbf{W}_{fx} \in \mathbb{R}^{s_h,s_d}$ are learned weighting matrices, $h_{t-1} \in \mathbb{R}^{s_h}$ is the hidden state vector at time step t, $x_t \in \mathbb{R}^{s_d}$ is the input vector at time step t and b_f is a bias. An intuitive example of the forget gate layer usage in NLP is when the gender of the subject of the sentence changes – in that case, the LSTM cell should forget about the old one.

The "second gate layer" is formed by two perceptrons - one with a sigmoid activation function and the other one with a hyperbolic tangent activation function. Using the perceptron with a hyperbolic tangent activation function a vector of so-called "candidate values" $\tilde{c}_t$ is computed. "Candidate values" determine which of the newly received information might be good to remember by the cell state:

$$\tilde{c}_t = \tanh(\mathbf{W}_{ch} h_{t-1} + \mathbf{W}_{cx} x_t + b_c)$$

$\mathbf{W}_{ch} \in \mathbb{R}^{s_h,s_h}$ and $\mathbf{W}_{cx} \in \mathbb{R}^{s_h,s_d}$ are learned weighting matrices and b_c is a bias. The "candidate values" vector picks the important parts of the information; however, it does not decide how much the particular part of the information is important. As a result, the "second gate layer" computes vector i_t using the perceptron with a sigmoid activation function which decides on how much the specific part of the information is important:

$$i_t = \sigma(\mathbf{W}_{ih} h_{t-1} + \mathbf{W}_{ix} x_t + b_i).$$

$\mathbf{W}_{ih} \in \mathbb{R}^{s_h,s_h}$ and $\mathbf{W}_{ix} \in \mathbb{R}^{s_h,s_d}$ are learned weighting matrices and b_i is a bias.

The general idea behind the "second gate layer" is to pick important parts of the information including expressing the level of their importance and adding those parts to the cell state vector. Example from the field of NLP: if the gender in a sentence changes, the second gate layer should pick out the new gender and store information about it in the cell state.

In the next steps the LSTM cell updates the current cell state vector c_t using the f_t, i_t and $\tilde{c}_t$ vectors computed in the first two gate layers and the previous cell state vector c_{t-1}[30]:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

With the cell state vector updated, the cell can now produce an output. The final output hidden state h_t is a filtered version of the cell state (pushed between -1 and 1 by a hyperbolic tangent function) based on current input and previous hidden state:

$$o_t = \sigma(\mathbf{W}_{oh} h_{t-1} + \mathbf{W}_{ox} x_t + b_o),$$

$$h_t = o_t \odot \tanh(c_t),$$

where $\mathbf{W}_{oh} \in \mathbb{R}^{s_h, s_h}$ and $\mathbf{W}_{ox} \in \mathbb{R}^{s_h, s_d}$ are learned weighting matrices and b_o is a bias.

For the sake of simplicity, all the transformations have been shown with input batch size equal to one. In real-world usage, the batch size s_b would be bigger, and input, hidden state and inner state vectors would become matrices with s_b columns.

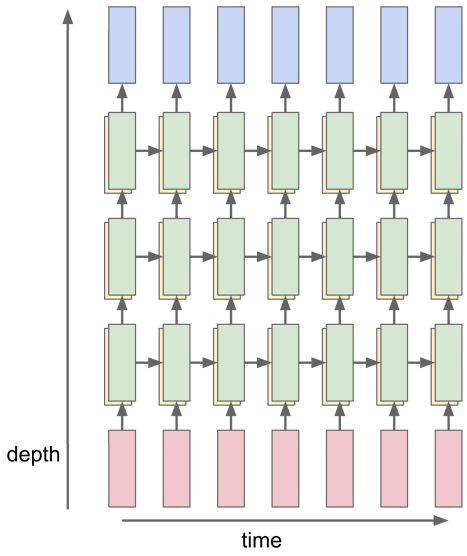
3.1.2 Multi-layer RNN

As it has been shown in other fields of machine learning (e.g., computer vision [25]), deep neural networks tend to perform better on various tasks – due to their higher ability to adapt to the data. Unlike feedforward neural networks, recurrent ones can be already considered deep [26] by their definition. If the recurrent neural network gets unrolled in time, it consists of multiple perceptron applications stacked on top of each other. However, there exist ways to make recurrent neural networks deep in different manners.

Multi-layer stacked recurrent neural networks have shown superior results to the single layer ones in the field of NLP [27]. Multi-layer recurrent neural network architectures, often used in language models which will be introduced later in this work, use simple passing of the hidden state to the next layer (Figure 3.4). The hidden state of the first layer is passed as the input to the

[30] Operation $\odot$ is defined as an element-wise multiplication.

Figure 3.4: Visualisation of the multi-layer recurrent neural network. Source: [18].



second layer cell at each time step. This goes for as many layers as required, and the final output is the output from the last layer at each time step. The multi-layer model architecture can be used with traditional RNNs as well as with more advanced LSTMs. However, in contrast with feedforward neural networks, only a small number of layers (usually two or three) is used in multi-layer recurrent neural networks in the field of NLP [27].

3.2 Convolutional neural network (CNN)

The following explanation is inspired by computer vision course at Stanford University [28]. Convolutional neural networks are mostly known from the field of computer vision. They have achieved excellent results [25] in computer vision tasks such as object detection and object classification. The general idea behind convolutional neural networks is to learn to focus on certain important features of the input matrix (i.e., the image) and create a new set of higher level features in each of the following layers. Intuitively, if the CNN is processing an image of a car, it might learn to recognise lines which correspond with the car body panels in the first layer and then in the next one it might learn to recognise wheels. In the further layers, it could learn to recognise the car's headlights or, for example, doors.

The architecture of the convolutional neural network consists of three types of layers: convolutional layer, pooling layer and fully connected layer. CNN architectures tend to be very deep since every new convolutional layer makes

it possible to recognise a new higher level set of features. However, for this work and the NLP algorithms which will be introduced later, explanation of a network with only a single convolutional layer and with input in the form of a simple two-dimensional matrix of real numbers will be provided.

3.2.1 Convolutional layer

The convolutional layer uses so-called filters to recognize important regions of the input matrix. Filters are learned weight matrices and their dimensions are hyper-parameters of the model.

Doing a convolution means sliding the filter matrix over the input matrix while calculating dot products of overlaying values of these two matrices along the way. Let for example $\mathbf{F}_1 \in \mathbb{R}^{2,2}$ be a filter, $\mathbf{I}_1 \in \mathbb{R}^{5,5}$ be the input matrix and $s_s = 1$ hyper-parameter called stride. Starting with top left corners of both matrices aligned on top of each other, the algorithm will be moving the filter's top left corner always by s_s positions at first in the horizontal direction and after reaching the end of the row it will move the filter to the beginning of the row which is s_s positions away from the current row and continue there as long as the whole filter matrix fits inside of the input matrix boundaries. This convolution will end up creating a new matrix $\mathbf{O}_1 \in \mathbb{R}^{4,4}$ consisting of the calculated dot products called the activation map (Figure 3.5).

The activation map shows how much in each filter sized region of the input matrix are the data similar to the learned filter weights. If this was a computer vision task, the filters could be learned to recognise for example vertical lines. Parts of the input image where there were vertical lines would result in higher values in the activation map.

Generally, having the input matrix $\mathbf{I} \in \mathbb{R}^{s_{im},s_{in}}$, filter matrix $\mathbf{F} \in \mathbb{R}^{s_{fm},s_{fn}}$ and stride s_s (s_s, s_{fm} and s_{fn} are hyper-parameters) results in activation map matrix $\mathbf{O} \in \mathbb{R}^{s_{om},s_{on}}$ where

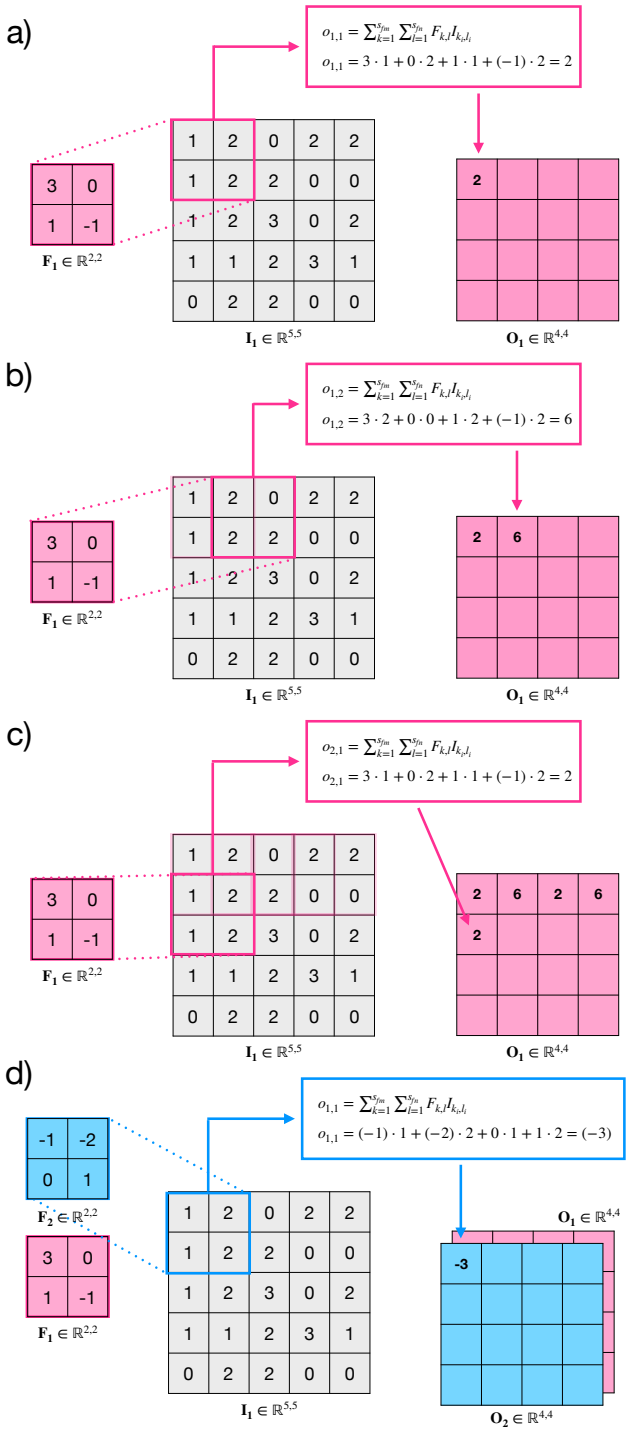
$$s_{om} = (s_{im} - s_{fm})/s_s + 1$$

and

$$s_{on} = (s_{in} - s_{fn})/s_s + 1$$

after doing the convolution. Let $f_{a,b}$ be the value in a-th row, b-th column of the matrix $\mathbf{F}$. First step of the convolution is to place $\mathbf{I}$ and $\mathbf{F}$ ($s_{fm} < s_{im}$ and $s_{fn} < s_{in}$ is assumed) on top of each other so that $i_{1,1}$ and $f_{1,1}$ values are overlaying. Then the filter matrix $\mathbf{F}$ is moved over the matrix $\mathbf{I}$ from left to right by s_s positions and at each step the activation value is calculated and stored into resulting activation map $\mathbf{O}$. At each step all the values of filter matrix $\mathbf{F}$ must be overlaying with corresponding values of $\mathbf{I}$. If $f_{1,s_{fn}}$ does not overlay any value from $\mathbf{I}$ the filter matrix $\mathbf{F}$ is moved along

Figure 3.5: Visualisation of a convolution example: a) first step of creating activation map $\mathbf{O}_1$ using filter $\mathbf{F}_1$, b) second step of creating activation map $\mathbf{O}_1$ using filter $\mathbf{F}_1$, c) fourth step of creating activation map $\mathbf{O}_1$ using filter $\mathbf{F}_1$, d) first step of creating another activation map $\mathbf{O}_2$ using a different filter $\mathbf{F}_2$. Image created by the author of this thesis.



the **I** to the beginning of **I** row $r = current\_row + s_s$. If $f_{s_{fm},1}$ does not overlay any value from **I** the algorithm stops. Activation value at each time step is calculated as

$$\sum_{k=1}^{s_{fm}} \sum_{l=1}^{s_{fn}} F_{k,l} I_{k_i,l_i},$$

where k_i and l_i are corresponding positions to k and l in the under-laying matrix **I**.

In the case when the stride and filter size parameters would be set so it would be impossible to move the filter matrix across all values in the input matrix, so-called padding would be applied. Padding means adding values around the input matrix to make it bigger and to enable the algorithm to run convolution over it. Typically zeros are added around the input matrix which is called zero-padding.

3.2.2 Pooling layer

In the deep convolutional networks used in the field of computer vision, typically many convolutional filters are applied at each convolutional layer. That tends the activation maps to be deep. As a consequence, the computational complexity gets higher with more applied filters.

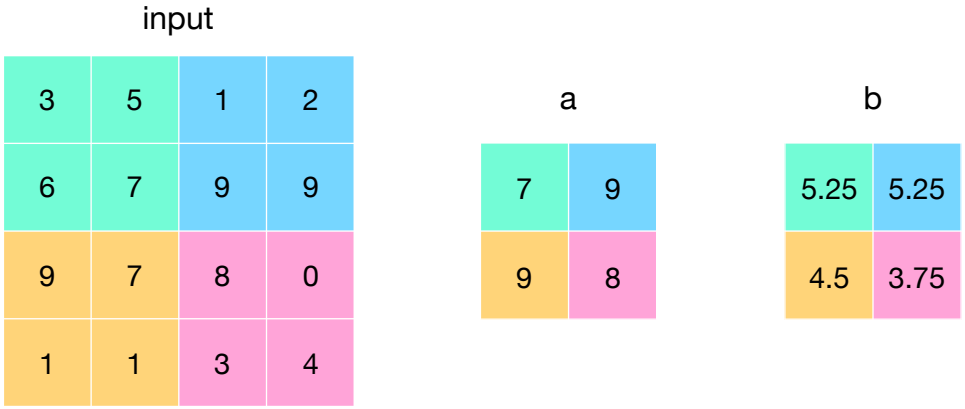
Pooling layers are designed to downsample the activation maps spatially but to keep their depth. The activation map gets downsampled by running a "pooling window" over the activation map matrix and calculating a certain function value over the currently "highlighted" input matrix values and saving these results into a new downsampled activation map. "Pooling window" size and the stride are hyper-parameters – in practice 2 by 2 "windows" with a stride of 2 are often used (resulting activation map is half the original size spatially). Commonly used functions to calculate a new single value from the whole currently "highlighted window" are maximum and average of the values.

Pooling layers are placed between convolutional layers. They proved to be efficient in terms of reducing computational complexity and can be seen as regularization elements in the network to prevent it from over-fitting. However, in the algorithms introduced later in this work, only an effortless max operation will be used in the convolutional network instead of the complex one which uses the previously described sliding "windows".

3.2.3 Non-linearity layer

Although the pooling layer acts as a non-linearity in the network, the convolutional neural network might use a separate non-linearity layer. It takes the activation map from the previous convolutional layer as the input

Figure 3.6: Visualisation of an example of the pooling layer with 2 by 2 sized "windows" and stride of 2. Input is the activation map from the previous layer. In this example there are two types of pooling layers: a) max pooling layer and b) average pooling layer. An image by the author of this work.



and applies a non-linearity individually to every value of the map. Often a rectified linear unit (ReLU) is used in convolutional networks. This non-linearity makes every negative number zero which leads to forgetting information (activations) which are not that important.

3.2.4 Fully connected layer (FC)

Although fully connected layers will not be used in any of the language modelling algorithms introduced later in this work, they should be briefly described to the reader to complete the explanation of basic convolutional neural networks.

Neurons in the convolutional layers are always connected to a specific area of the input matrix. As a consequence, they cannot be used to describe the input matrix (i.e., image) as a whole[31].

To make it possible to make decisions (i.e., classify object in an image) about the entire input matrix, fully connected layers have been introduced. Neurons of these layers are connected to all values (activations) from the previous layer. Fully connected layers are equivalents to hidden layers in basic neural networks.

[31] Assuming the filter is smaller than the input matrix. If the convolutional filter would be the same size as the input, the convolutional layer would become fully connected. As a consequence, the convolutional layer can be converted to a fully connected one. Moreover, the fully connected layer can also be transformed into a convolutional layer resulting in a large matrix with mostly zeros.

The fully connected layer is usually used as the last layer of convolutional neural network architecture, so it can take high level extracted features as its inputs and output class predictions (typically by utilising the softmax activation function). state-of-the-art convolutional neural networks [25, 29] from the field of computer vision often use more fully connected layers at the end of their architectures.

3.3 Highway network

Deep neural networks suffer from vanishing gradient related problems since the activation function used in neurons of a deep neural network squishes the values into a small number range (e.g., the sigmoid's range of values is between 0 and 1). As the network gets deeper, these activation function output values tend to get small (vanishing gradient) or big (exploding gradient). Exploding gradient can be solved by multiple proposed methods (e.g., [30]). The vanishing gradient cannot be solved once it happens; the network has to be prevented from it. Moreover, due to the computer representation of floating point numbers, these small values can become zeros.

As a result, a significant change in the input generates only a small change in the activation functions down the stream. Calculating gradients from these very small or zero values may end-up in zero gradients making it hard to train the network. Unlike exploding gradients which can be prevented by applying, for example, so-called "gradient clipping" (proposed in [30]) to the network, vanishing gradients are harder to be dealt with.

One of the possible solutions to this problem which has been proposed in [31] is called the Highway network, and its overall idea is similar to the gating mechanism used in already explained Long short-term memory networks [23].

In a layer of a simple neural network is applied a non-linear transformation

$$y_n = H(\mathbf{W}_H x + b_H),$$

where H is a non-linear function (e.g., sigmoid), vector $y_n \in \mathbb{R}^n$ is the output of the layer, $\mathbf{W}_H \in \mathbb{R}^{n,n}$ is a learned weighting matrix, the vector $b_H \in \mathbb{R}^n$ are learned biases and $x \in \mathbb{R}^n$ is a single vector input to the layer.

Highway networks enable each layer to learn which parts of the information are important and keep those without further unnecessary transformation – a parallel with the gating mechanism in the LSTM. Highway networks introduce additional so-called "transform gate" non-linear transformation

$$y_T = T(\mathbf{W}_T x + b_T)$$

where T is a non-linear function, $\mathbf{W}_T \in \mathbb{R}^{n,n}$ is a learned weighting matrix and $b_T \in \mathbb{R}^n$ are learned biases. The output vector y_T decides how much a certain part of the input information should be transformed. On contrary, the so-called "carry gate"

$$y_C = (\mathbf{1} - y_T) \odot x$$

is the exact opposite of the "transform gate" and decides how much the input information will be passed without further transforming. Here $\mathbf{1} \in \mathbb{R}^n$ is a vector of ones.

The output from the highway network's layer is defined by putting carry and transform gates together so

$$y = y_T + y_C =$$

$$= T(\mathbf{W}_T x + b_T) + (\mathbf{1} - T(\mathbf{W}_T x + b_T)) \odot x$$
$$= T(\mathbf{W}_T x + b_T) + x - T(\mathbf{W}_T x + b_T) \odot x.$$

Although the main usage of the highway network takes place in very deep networks to prevent them from issues related to the vanishing gradient, in one of the machine learning algorithms explained later in this work a single layer of the highway network will be used in a shallow neural network, to make it able to learn additional information from the input [32].

Chapter 4

State-of-the-art methods of natural language processing

In this chapter, the state-of-the-art methods for natural language processing with a significant focus on modelling polysemy will be presented. Some later explained models may not be used on their own and need to be implemented as a part of a bigger language model – these models will be mentioned bellow so the reader can get familiar with individual parts of the resulting state-of-the-art models. General introduction of this chapter has been inspired by a great review of recently proposed methods in [27].

As the amount of textual data available on the Internet has grown, new sophisticated and robust language models learning probability distributions over particular languages on huge corpora has been proposed. Today's state-of-the-art language models are much more capable than just predicting the next word in a sentence.

They can predict which sentences would be grammatically correct, however, unlikely to appear given the context. Furthermore, they can understand long term dependencies between words[32] as well as understand the text itself so well they can correctly answer questions asked in natural language based on the input text.

Most of the modern language models are based on dense vector word representations and neural networks, especially the recurrent neural networks in their long short-term memory form. Long short-term memory networks

[32] An example of a long term dependency in an article could be a situation when there is a subject mentioned at the beginning of the article and at multiple following positions in the article some attributes are given to the subject. A modern and well-trained language model should be able to encapsulate all these attributes to the word representation.

bring the capability of remembering complex information about a particular subject in the sentence. Also, recurrent neural networks, in general, are an intuitive way to learn information from sequential data such as sentences of natural language.

4.1 Character level convolutional neural network

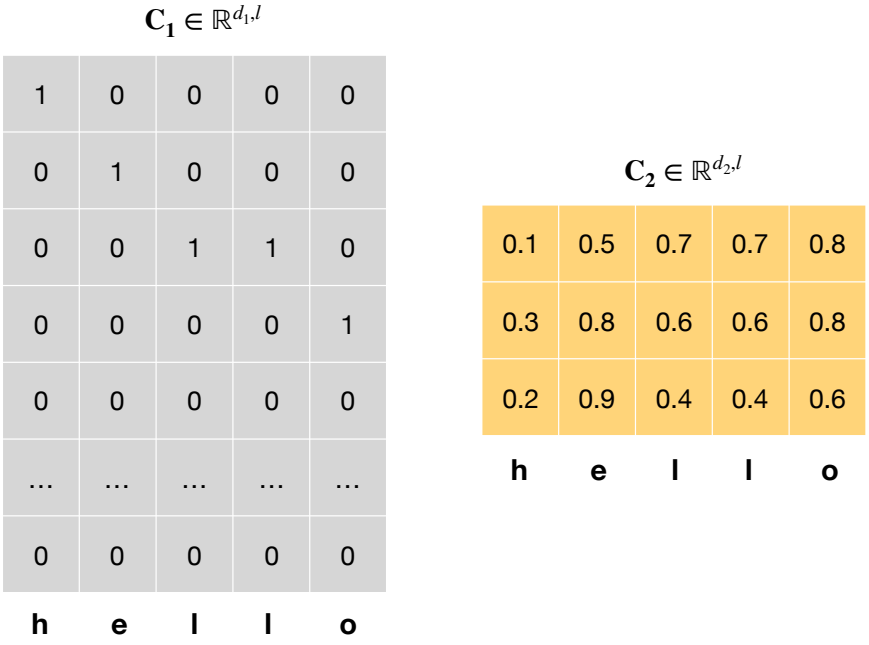
Neural language models usually use word embeddings as their inputs. However, this might be an issue in languages with a rich morphology (e.g., Czech) since each morphological word form would be represented by a distinct word embedding (there could be tens of word forms per each lemma). As a consequence, the later neural network would be dealing with high-dimensional data which would lead to worse performance. Furthermore, the neural network probably would not be able to understand the similarity between rare word forms which are based on the same lemma. This issue is usually solved by applying a morphological analysis to the input words. However, a new more general and inherently unsupervised approach based on convolutional neural networks has been proposed in [32]. The following explanation of the approach is based on this paper.

The proposed solution is connected to the input of a neural language model as a word preprocessor (from words to embeddings). It uses character-level input and creates word embeddings of a fixed dimension which is determined by the number of filters used in the under-laying convolutional network. The solution enables modelling information about morphemes in a word without any linguistic knowledge. It is also very robust when processing misspelt words.

First of all the vocabulary of all possible characters as well as their character embeddings of the dimension d must be created. The character embeddings can be fixed (for example one-hot encoded) vectors or can be learned as low dimensional dense vectors. The authors of the paper suggest using low dimensional dense vectors (learned along with the rest of the model) as the embeddings which have shown to perform slightly better according to the paper.

Having established the character embeddings, the algorithm splits the input sentences into individual words and then, again, splits these words into separate characters. Each character is then represented by its embedding vector created in the previous step. These character embedding vectors of each word are concatenated together which results in a matrix $\mathbf{C} \in \mathbb{R}^{d,l}$ (Figure 4.1) where d is the dimension of the character embeddings and l is the length of the longest word in the corpus per each word. For words shorter than l, zero padding is used.

Figure 4.1: An example of the possible embeddings of characters. C_1 uses a one-hot encoding and thus d_1 is equal to the number of distinct characters in the language. On the contrary, C_2 uses learned dense vector character encoding which results, in this case, in $d_2 = 3$. The results of both should be similar. However, the authors of [32] suggest that using dense representation performs slightly better. Therefore, only the dense representation will be taken into account in the later explanation. In this particular example, there is not a longer word than "hello" in the language. This visualisation has been created by the author of this thesis.



One of the model's hyper-parameters is the number of convolutional filters h to be learned during the training process. These filters can be of different sizes, their widths $w_1, w_2, ...w_h$ are hyper-parameters of the model as well, and their heights are always equal to the dimensionality of the character embeddings d.

All the learned convolutional filters $\mathbf{H}_1, \mathbf{H}_2, ..., \mathbf{H}_h \in \mathbb{R}^{d,w}$ are applied to the character embeddings matrix $\mathbf{C}$ using a narrow convolution (without zero padding, stride is one). As a result, h activation vectors $a_1, a_2, ..., a_h$ are calculated, and a non-linearity is applied individually to their values. Length of each of these vectors depends on the width of the particular convolutional filter (example in Figure 4.2). A filter of the width w is responsible for an activation vector h of the length $l - w + 1$. Intuitively, in this process the convolution picks some n-gram ($n = w$) of the input word and calculates

Figure 4.2: An example of the character-level convolutional network processing the input word "hello". In a: first step of the convolution using the filter H_1, in b: completed convolution using the first filter H_1 and first step of the convolution using the second filter H_2. C is the input character embeddings matrix, a_1, a_2 are the activation vectors created by applying a convolution using the filters H_1 and H_2 to the input matrix and then applying a non-linearity σ (in this case a sigmoid) to the results, $w_1 = 3$, $w_2 = 2$, $l = 5$ and $d = 3$ are given hyper-parameters. This visualisation has been created by the author of this thesis.

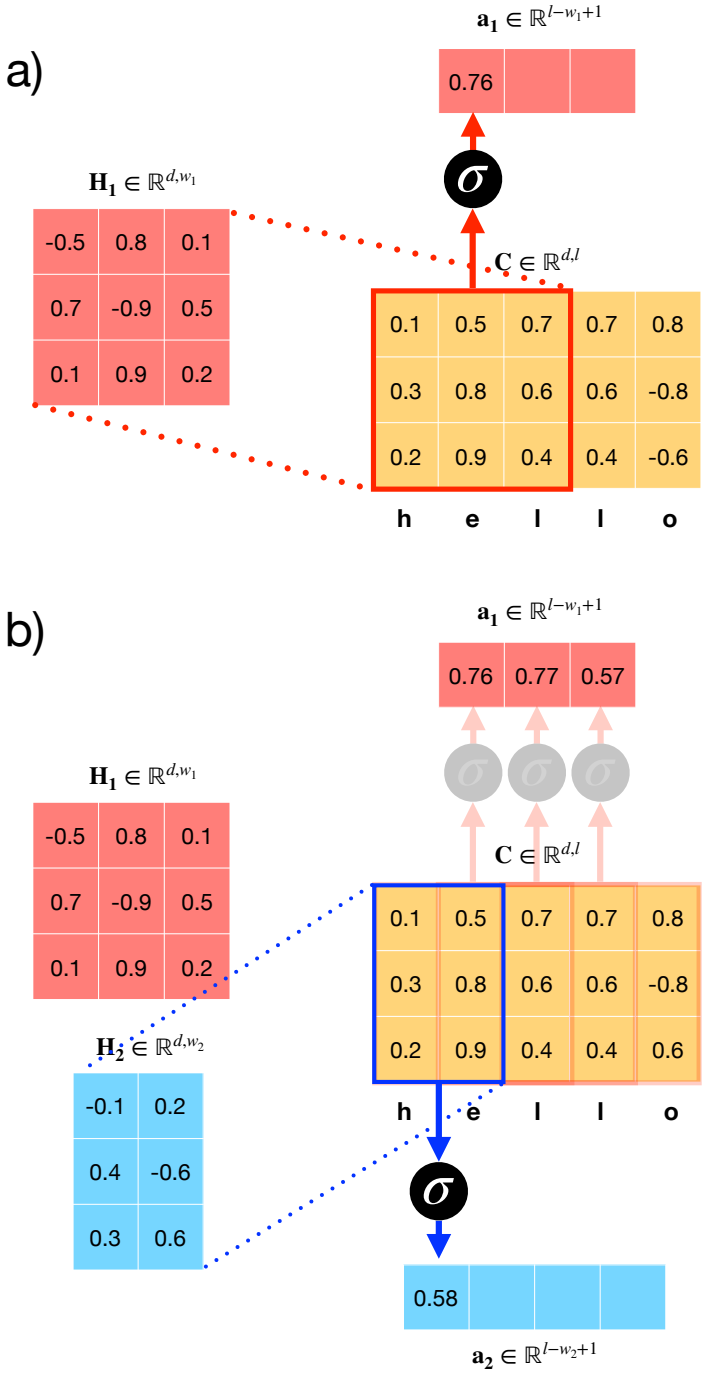


Figure 4.3: This visualisation follows Figure 4.2. It shows the extraction of maximal values from the activation vectors. According to the example, the activation vector a_1 chooses the tri-gram "ell" as the most important one and similarly the activation the activation vector a_2 does pick the bi-gram "lo"; y is the final feature vector. This visualisation has been created by the author of this thesis.

feature value of it this; can be seen as assigning importance to the particular n-gram[33].

Lastly, a max value of each of the activation vectors $a_1, a_2, ..., a_h$ is taken and put into a resulting feature vector $y \in R^h$ (Figure 4.3). Again, this intuitively means picking the most important (interesting) n-grams of each word. As a result, this character-level convolutional neural network can recognise the lemmas of words without any morphological knowledge about the language. Weights of the mentioned convolutional filters are trained together with the downstream language model using stochastic gradient descent and for example Hierarchical softmax [33] (word-level predictions).

4.2 Context-dependent bi-directional language model

A good language model should be able to precisely compute the probability of a sentence appearing in the input text. The general approach is to use the chain rule. That means to simply calculate the conditional probability of each word w_i where $i \in 1, 2, ..., n$ occurring in the sentence (i.e., a sequence of words $w_1, w_2, ..., w_n$) based on the previous words already seen in the sentence and then multiply these conditional probabilities:

$$p(w_1, w_2, ..., w_N) = \prod_{k=1}^{n} p(w_k | w_1, w_2, ..., w_{k-1}).$$

However, since essential dependencies in natural language can occur in both directions from the target word, it is intuitive to model the conditional

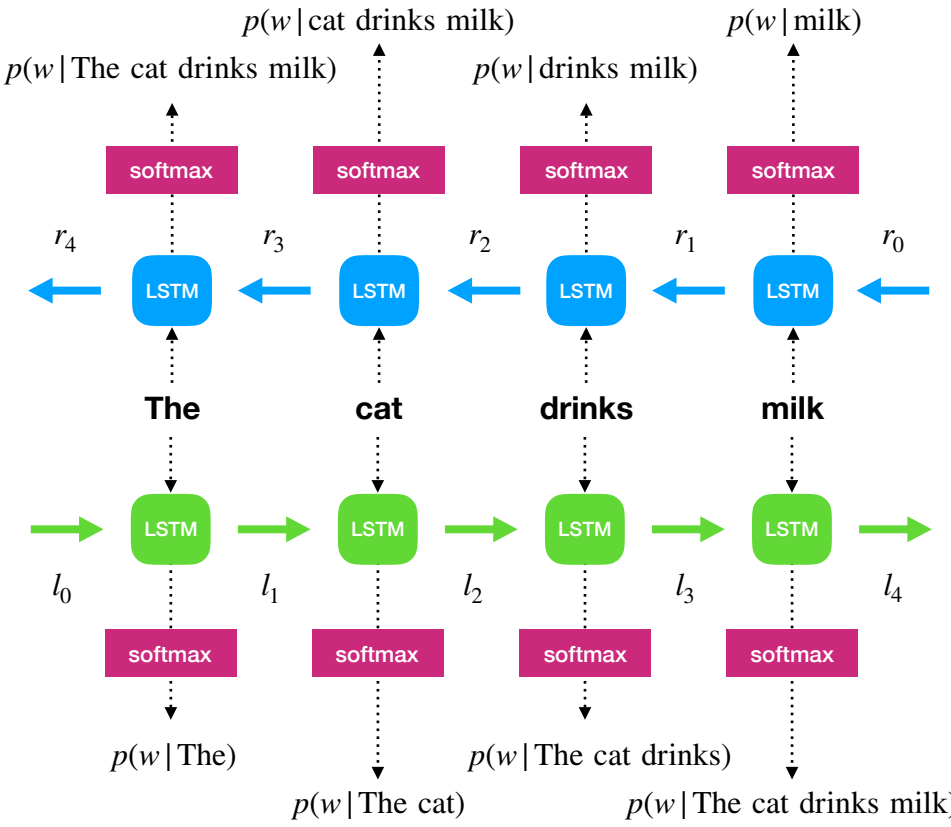
[33] For example if the word on the input would be "heeeeeeello" and the algorithm would be using filters of width $w = 3$, the "hee", "ell", "llo" should be triggering more attention to the convolutional filter then n-gram "eee" which does not carry much useful information.

probabilities in the right to left pass through the sentence as well:

$$p(w_1, w_2, ..., w_N) = \prod_{k=1}^{n} p(w_k | w_{k+1}, w_{k+2}, ..., w_{k-1}). \text{ [34]}$$

Because textual data are of a sequential type, the recurrent neural networks are an inherent solution to this problem. The long short-memory networks (LSTM), which are capable of remembering reasonably complicated dependencies over long sequences of data has shown great results in the field of natural language modelling as well.

Figure 4.4: Visualisation of the bi-directional language model using LSTM networks in both directions according to [34]; $l_1, ..., l_4$ are hidden state vectors of the forward LSTM network (green), $r_1, ..., r_4$ are hidden state vectors of the backward LSTM network (blue), w is any word from the dictionary or <start>/<end> tokens (placeholders expressing the start/end of the sentence). Figure created by the author of this thesis.



Modern bi-directional language models use two LSTMs – one for the forward (from left to right) and one for the backward (from right to left) pass through

the sentence. At every word position w_t the LSTM outputs the hidden state of the step t corresponding to one of the directions. By using those two LSTMs each in a different direction, two hidden states are computed at each step t. These two hidden states encapsulate information about the left as well as the right context of the word w_t (Figure 4.4).

Different models use different techniques to create single vector word representation of the two direction-dependent ones, but most often a simple concatenation of the vectors is used resulting in a single vector word representation encapsulating the word itself as well as its context in both directions. A Softmax layer is used to predict the next/previous word based of the hidden state of the LSTM at each step.

4.3 Word2vec

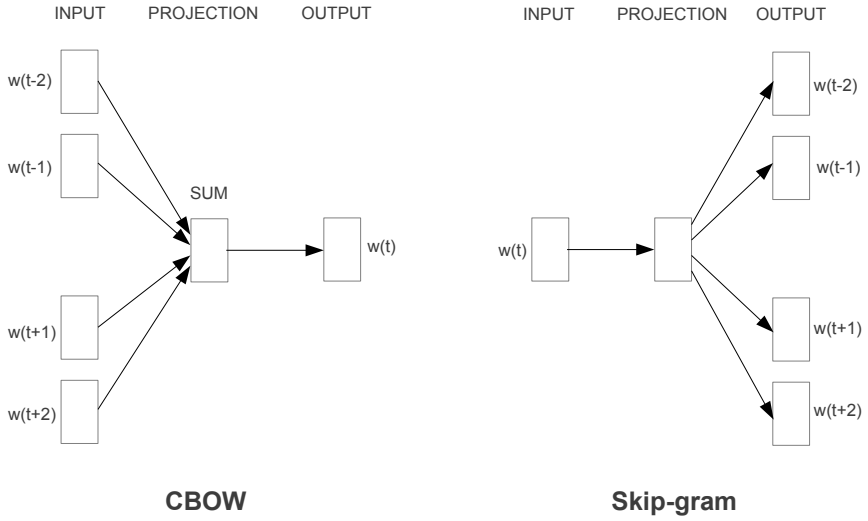
Word2vec neural language model which has been introduced by the paper [35] in 2013 is probably the most frequently used one even today. It has come up with an efficient way to learn distributed word representations. Furthermore, it has shown that by applying simple algebraic operations to the learned word representations, surprisingly complex meaning-based relationships between words can be modelled. Although Word2vec cannot deal with polysemy, it will be quickly explained in this thesis as a requirement for the understanding of the more advanced models. The following description is based on [35].

First of all, the Word2vec model prepares a dictionary of all the possible words in the training data. The input to the model is then positionally encoded (one-hot encoding). There are two possible architectures of this model: the Continuous Bag-of-Words model and the Continuous Skip-gram model (Figure 4.5).

The Continuous Bag-of-Words model (CBOW) architecture utilizes a single hidden layer of linear neurons and a Softmax output layer. The objective of the model during training is to predict the centre word based on the continuously distributed representation of the centre word's context.[34] The context is limited in size by a fixed window taking into account only some number of the centre word's neighbours in both directions. Since it is a "bag" of words, the surrounding context words are not ordered. This architecture has shown better results on smaller datasets due to its ability to smooth over the context information (the whole context is encoded in one representation, rare or misspelt words might be smoothed out).

[34] For example if the sentence in the training dataset would be "My dog is very fat.", at some point the Bag-of-Words model would be trying to predict the word "very" based on the context words "dog", "my", "is" and "fat".

Figure 4.5: A visualisation of the Word2vec architectures. CBOW model architecture on the left, Skip-gram model architecture on the right. Visualisation borrowed from [35].



On the contrary, the objective of the Skip-gram model architecture is to predict the context given the centre word. The model tries to predict the surrounding words within a fixed range during the training. It has been shown in the paper that predicting broader context improves the quality of the learned word vectors but increases the computational complexity. To model the fact that more distant words are usually less important to the centre word, weighting can be applied during the training process.

Both the models mentioned above are used in the same manner. When the model is trained (to predict the centre words or the contexts), its hidden layer weights are extracted. Since the input, as well as the labels, are one-hot encoded, it means the weights of each linear neuron in the hidden layer are corresponding to particular words in the same order they were represented by the one-hot vector. As a result, in each of the linear neuron, there are v weights (v is the size of the vocabulary). By concatenating weights from all of the neurons of the hidden layer a matrix $\mathbb{W} \in \mathbf{R}^{v,d}$ where d is the number of neurons is obtained. In this matrix, each row is a word embedding of the positionally corresponding word in the vocabulary.

Word2vec learned word embedding vectors have shown great results in terms of modelling similarity between words. By simply applying basic algebraic operations to them, surprisingly precise and deep relationships can be discovered. For example

$$\text{vector(Berlin)} = \text{vector(Paris)} - \text{vector(France)} + \text{vector(Germany)} \text{ or}$$

$$\text{vector(smallest)} = \text{vector(biggest)} - \text{vector(big)} + \text{vector(small)}.$$

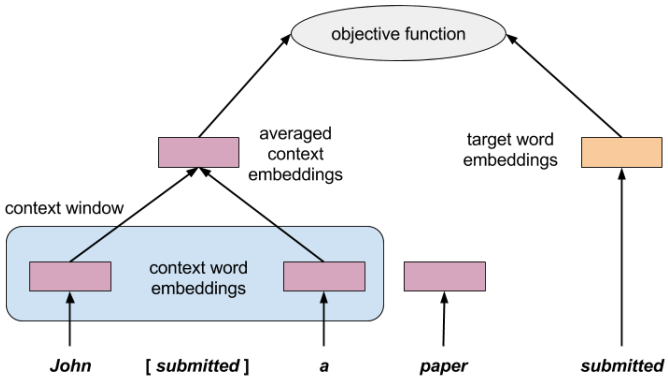
4.4 Context2vec

The Word2vec model presented in the previous section has shown good results on the language modelling task. However, there are few serious drawbacks with this model. Due to its fixed word representation where each word is assigned a single dense vector, it is not capable of modelling different senses of the same word based on the context (polysemy). Moreover, the fixed size window region of words around the target word which are analysed during the training process may lead to bad understanding of the context, since important words (i.e., subject of the sentence) can be placed further away from the target word outside of the fixed window region. Also, representing words surrounding the target as an unordered set of equally relevant words cannot understand the context deeply. One of the possible solutions to these problems has been introduced in [36] as a so-called Context2vec model. The following description is based on this paper.

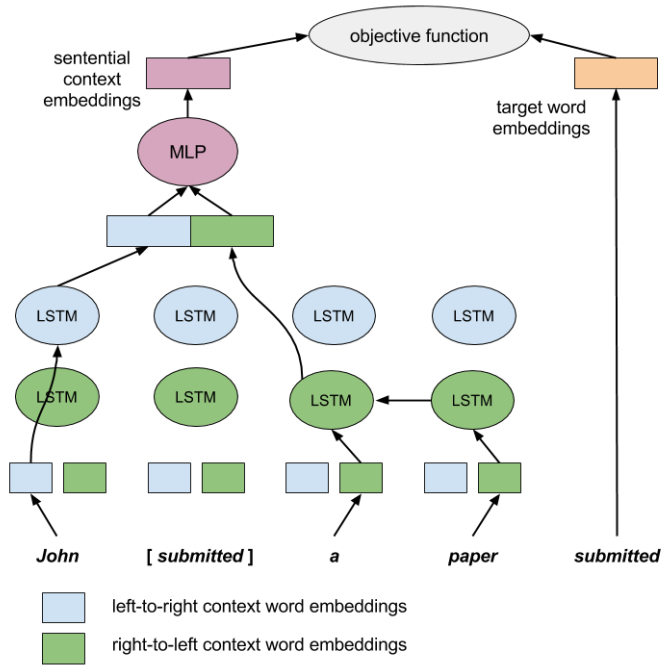
The Context2vec model is heavily inspired by Word2vec's CBOW model. It uses almost the same architecture (see comparison in Figure 4.6), negative sampling [35] and the same way of training as Word2vec. However, it replaces the part where Word2vec uses fixed size window to encapsulate information about the context surrounding the target word into a single vector by a much more advanced bi-directional recurrent long short-term memory network (presented in the Theoretical background chapter of this thesis). Using long short-term memory network to gain knowledge about the word's context inherently leads to a much deeper understanding of complex dependencies in the sentence in comparison with the simple approach used in Word2vec.

It is important to mention that Context2vec uses the bi-directional recurrent neural network in a rather unusual way in comparison with other language modelling algorithms. It splits the input sentence into two parts by the target word resulting in a sequence of words l before and a sequence of words r after the target word. The target word itself is not passed to the recurrent neural network at all. For the sequence l the hidden state of the last word from this sequence computed by forward (left-to-right) pass through the LSTM over the sequence l is produced. The same goes for the sequence r, however, in this

Figure 4.6: A comparison of the Word2vec (a) and Context2vec (b) models. Visualisation borrowed from [36].



(a) word2vec *CBOW*



(b) *context2vec*

case, it produces the hidden state of the first word in the r sequence computed by the backward pass (right-to-left) throught the LSTM over the r sequence. As a result, two hidden state vectors each describing one of the two directions of the context are produced.

To fit the architecture used in the Word2vec model, the Context2vec model utilises a multilayer perceptron to pick important features from the left and the right context vector representations produced by the bi-directional LSTM. In the result, a single sentential context embedding is produced. This embedding is used in the training phase in the same manners as the averaged context embedding in the Word2vec model.

After training, the weights of the multilayer perceptron are frozen and used for predictions of context embeddings of new sentences which is the objective of the Context2vec model. Of course, based on the context embeddings the most probable word occurring in the context can be predicted by using a Softmax layer.

The Context2vec model is capable of modelling contextual dependencies as well as modelling polysemy (which is important for the objective of this thesis). Unfortunately, this model is hardly usable for languages with a rich morphology since it does not use any module for morphological analysis. Overall, the Context2vec model has shown good results on sentence completion, lexical substitution and word sense disambiguation task according to the paper.

4.5 Deep contextualised word representations (ELMo)

Embeddings from Language Models (ELMo) which have been presented in [34] have been one of the most important innovations in the field of natural language processing. Not only that the model which has been introduced this study is capable of state-of-the-art accuracy language modelling, but it also brings transfer learning to the NLP. However, for this thesis, the essential property of the model behind ELMo is its ability to inherently distinguish between distinct word senses (it is capable of modelling polysemy). This section is based on [34]. The architecture of the model behind the Embeddings from Language Models will be called for simplicity “ELMo” in this thesis from now on.

The general idea behind ELMo is to train a sophisticated language model on a large corpus and then allow the downstream task developers to fit the robust and complex ELMo word embeddings to the needs of their supervised models without the necessity to do the computationally expensive language model training from scratch on their own. This has brought the transfer-learning into the field of natural language processing, making it possible for smaller developers without the access to an adequate computational power or

in-depth knowledge of machine learning to implement state-of-the-art language processing techniques into their applications.

ELMo works remarkably well even on languages with rich morphology. It does not use any morphological analysis tool or a human-made dictionary of morphemes. Instead, ELMo takes use of the character level convolutional neural network (charCNN) which has been previously presented in this chapter. The input words are split into individual characters and passed to the charCNN which computes context-independent word embeddings. These embeddings represent words based on the same lemma close to each other in the vector space and do not get confused by misspelt words.

In the next step, ELMo passes the context-independent character-level word embeddings through the 2-layer highway network (explained in the Theoretical background chapter of this thesis) where interactions between different n-grams of the word can be captured. The output of the 2-layer highway network is used as an input to the bi-directional language model.

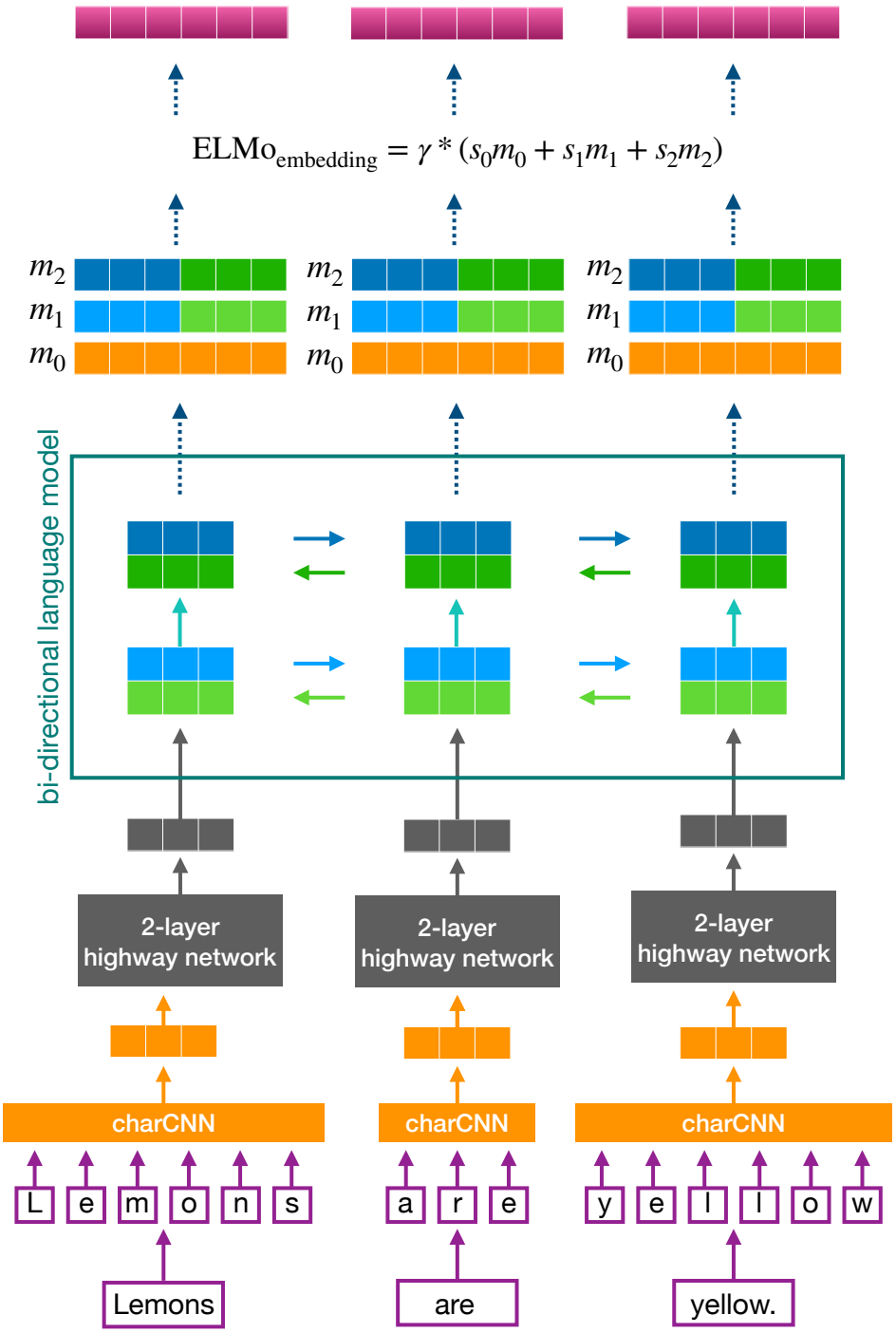
The bi-directional language model utilised in ELMo consists of two layers [35] of LSTM network in both directions. The inputs to the first layer of the LSTM network are the context-independent word embeddings from the character level convolutional neural network. For each word, four hidden states of the LSTM network are computed – two from both-directional passes of the first bottom layer and the other two similarly from both directions of the second upper layer. It has been shown in the paper that the bottom layer of the network models syntactic dependencies whereas the upper layer focuses more on semantic aspects of the context.

Where ELMo differs from previous studies is the way it handles the output (hidden state) vectors from the bi-directional language model. At first, ELMo concatenates forward and backward pass hidden states at each time step and in each layer into a single vector. As a result, two concatenated vectors (one vector per layer of the LSTM network) are computed per each word. The concatenated vector m_1 from the first layer has shown to describe the syntactic level information whereas the concatenated vector m_2 from the second layer of the LSTM focuses on the semantic meaning of the whole context. To add sub-word character-level information ELMo takes the output from the character level convolutional neural network and concatenates it with itself (to get a vector of the size of m_1 or m_2) into a single vector m_0.

The three vectors m_0, m_1, m_2 are the outputs of the bi-directional language model for a particular word of the input sentence. To create a task specific

[35] The output from the first layer is passed as an input to the second layer of the LSTM network at each time step.

Figure 4.7: Visualisation of the architecture of the model behind the Embeddings from Language Models (ELMo): s_0, s_1, s_2 are learned softmax-normalised weights and $\gamma \in \mathbb{R}$ is a learned weight involved for optimisation purposes. This visualisation has been created by the author of this thesis.



ELMo word embeddings, the downstream task developer uses a linear combination of the three vectors

$$\text{ELMo}_{\text{embedding}} = \gamma * (s_0 m_0 + s_1 m_1 + s_2 m_2),$$

where s_0, s_1, s_2 are learned softmax-normalised weights and $\gamma \in \mathbb{R}$ is a learned weight involved for optimisation purposes; $m_0, m_1, m_2 \in R^h$, where h is number of filters in the character level convolutional network, are ELMo vectors.

To sum it up, the bi-directional language model in ELMo is trained on a huge language corpus. Afterwards, its hidden state vectors are frozen. Downstream task developers initialize their ELMo model with the language corresponding frozen weights. As a result, they receive three word describing vectors per each word in a sentence. Although these vectors could be used on their own, to take full advantage of ELMo, a linear combination of these vectors is trained alongside with the downstream supervised model to compute the ELMo word embeddings. Such embeddings are capable of fitting to individual needs of the downstream task by using the right amount of character-level, syntactic and semantic information.

Chapter 5

Realisation

The objective of this thesis is to apply the state-of-the-art methods introduced above on the Czech word sense disambiguation task. The model should be able to decide if two use cases of a word with the same lemma are expressing the same meaning (word sense) or not. Furthermore, unsupervised clustering approach could be theoretically able to find all the word sense classes existing in an input textual data. In this chapter, different possible approaches to this task will be presented, and their results will be analysed. Moreover, the essential process to acquire relevant Czech language training data will be described.

5.1 Acquiring relevant Czech textual data

There are two different types of Czech language textual datasets which will be used in this thesis. To use supervised methods to disambiguate between different word senses a labelled dataset of words including the class identification of the particular word sense must be acquired. On the other hand, to train a language model, a huge language corpus must be used. In this section, the used methods for obtaining and processing these data will be described.

5.1.1 Czech language corpus

There are multiple Czech language corpora available for free for educational purposes. They differ in quality and the source of the data. Some use the publicly available data from the internet (sometimes including comments, captions and other textual data from social networks) and some are solely based on printed publications. In this thesis, I have been using the SYN2015 [37] corpus published by the Institute of the Czech National Corpus, Charles University in Prague.

lemma	sense_id	example
bank	1	I have got much money in the bank.
bank	2	These animals live on the banks of the river.
car	3	Volkswagen Beetle is the ultimate car!
bank	1	To buy a Beetle, we need to rob the bank!

Table 5.1: An example of the desired word sense disambiguation dataset format. The field sense_id contains the word sense identification according to the particular example.

The SYN2015 corpus contains 100 million words in 8 004 732 sentences extracted from 3 376 printed publications from 2010 to 2014 only. Sentences from these publications are shuffled in the corpus due to the copyright law. However, every sentence is assigned to its original publication. The corpus contains almost equally text from three following categories: newspapers and magazines, fictions and non-fiction. For each word, the corpus contains a lemmatised version of it and adds a rich morphological as well as a syntactical annotation. Unfortunately, it does not offer a sufficient amount of information in terms of word sense classification. Hence it will be used only as a training corpus for a language model in this thesis. [37]

The corpus comes in a huge 10GB XML formatted file. To use the corpus for the training of a language model, individual sentences need to be extracted from this XML file. Due to its enormous size, conventional XML parsing libraries for Python could not be used without further modifications. As a result, I have written a Python script (`syn2015/parser.py`) implementing a simple automaton to parse out individual words and concatenate them into sentences. A much smaller data file (less than 700MB) containing a sentence per each line has been produced by running this code on the SYN2015 corpus. Such data file can be easily used for training of a language model.

5.1.2 Czech word sense disambiguation dataset

At the time of the creation of this thesis, there were no explicit word sense disambiguation datasets for the Czech language available. Such dataset should contain a large amount of distinct word lemmas each provided with multiple examples of different usages in a sentence with the additional information about the word sense class used in the particular example (see Table 5.1).

Because of the missing Czech dataset, I have had to create custom dataset based on available online Czech dictionaries. However, most of the online Czech dictionaries do not offer enough examples of different word sense usages or do not have the examples at all.

The first Czech online dictionary that have come to my mind is the *Internetová jazyková příručka* [38] by Czech Language Institute of the Czech Academy of Sciences which contains some word usage examples, but does not describe the individual word senses at all, hence is not usable for this task.

Slovník spisovného jazyka českého [39] is another online dictionary published by the same institution. This dictionary contains most of the data which are needed for the supervised word sense disambiguation model training, but unfortunately, it is hardly processable by a computer. The format of this dictionary is the same as the one used in the printed version of the dictionary. To save space it uses many abbreviations in the notations as well as in the example sentences. Processing such a dictionary by a computer would be really hard and probably not accurate.

Another Czech online dictionary worth mentioning is *The Valency Lexicon of Czech Verbs with Complex Syntactic-Semantic Annotation* (valex) [40] developed at the Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University, Prague. This dictionary would be quite useful in this thesis if it was larger and was not containing only verbs. The other "officially" published dictionaries available on the internet usually do not contain the data necessary for this task or their format is not easily processable by a computer (for example *Příruční slovník jazyka českého (1935–1957)* [41] which returns digitalised printed dictionary image data).

Fortunately, the open community online dictionary Wiktionary [42] contains all the information needed to create a word sense disambiguation dataset for the Czech language. Although, this dictionary contains many mistakes (misspelt words, incorrect example sentences), it is still the best available source of these data online.

The whole Wiktionary (including 238 languages, 110 237 keywords) can be downloaded as an SQL import script or as an XML file. Since there was no easily readable documentation of the database scheme available, I decided to use the XML dump file. Sadly this XML file is not well structured, and Wiktionary itself says that it is not meant to be processed by a computer. The XML file contains all the relevant information inside a single XML tag as a plain text which was created by humans, so it tends to be corrupted.

I have come up with another script (`wiktionary/wiktionary_parser.py`) implementing an automaton processing the Wiktionary XML dump. This script takes the input XML file and for each Czech word finds all the possible meanings in the plain text data as well as all the examples of usages in sentences which are available. As a result, a tabular dataset corresponding to the format in Table 5.1 has been generated.

There has been 12 789 different Czech substantive and 6 926 adjective in-sentence usage examples found in the Wiktionary XML dump file. It is important to have at least two different word senses per word with at least one sense containing at least two different in-sentence usage examples to be able to learn to understand which usages are referring to the same meaning of the word and which are not.

Therefore I have created a dataset corresponding with the description above. This dataset contains only 2066 examples (includes adjectives and substantives) which means that most of the Wiktionary data are not suitable for achieving the goal of this thesis. Furthermore, I have found out that a lot of these examples are rude ones and not suitable for the academic environment at all (people were probably getting creative about words they found entertaining in some ways). However, since there are not better Czech word sense disambiguation data available, this dataset will be used from now on in this thesis.

5.2 Using the Context2vec model

The Context2vec model is not a widely used one and therefore it is hard to find good pretrained models for languages other than English. At the time of the creation of this thesis there was no pretrained model for the Czech language publicly available.

As a result, I have trained my Context2vec model from scratch using the implementation [43] from the author of the original paper [36]. I have decided to use the SYN2015 corpus as a training dataset for this model. Unfortunately, Context2vec model is very expensive to be trained in terms of the computational power as well as the memory, so I was not able to train it on the whole corpus.

I have been using Google Cloud to run these computations, and the bottleneck was the power of individual CPU cores while the Context2vec training algorithm was not able to use efficiently two or more cores in parallel. Although it could be much faster to train the model on a GPUs, the pricing of GPU cloud instances with a sufficient amount of RAM was not suitable for me, and sadly the university could not provide me with the needed computational power as well.

In the result, I was forced to make the corpus smaller by leaving out most of the sentences. In order to make the model trainable in my conditions, I have had to make the corpus about 100 times smaller than the original one resulting in a one million words dataset. Training the Context2vec model for 50 epochs using the one million words dataset took me more than a week of constant

sent. a		**sent. b**		**sent. c**	
inscenaci	(inscenation)	všechny	(all)	smečce	(gang)
metodě	(method)	některé	(some)	bali	(Bali)
knížete	(prince)	koupání	(bathing)	tenise	(tenis)
médium	(medium)	učitelky	(teachers)	remíze	(tie)
hry	(games)	nastávající	(oncoming)	výzvě	(challenge)

Table 5.2: Examples of the top five target word predictions by the Context2vec model. The target word position is annotated as "#". **sentence a**: "Po celou dobu studia gymnázia jsem se snažil mít co nejlepší známky, protože jsem doufal, že by mi to mohlo pomoci při následném studiu na pražské # škole." ("Throughout my studies at the grammar school, I was trying to get the best grades possible because I was hoping it would help me in my subsequent studies at a # in Prague") **sentence b**: "Koupil jsem si ojetý # značky Škoda a zpětně musím říct, že to byla dobrá koupě a to i přesto, že mě původně všichni odrazovali od onoho litrového tříválce." ("I have bought a used Skoda # and I have to say that it was a good purchase, despite the fact that I was initially discouraged by the one-liter three-cylinder engine.") **sentence c**: "Nedávno jsem se procházel po # a kochal jsem se vánoční výzdobou." ("I was recently walking around the # and enjoying the Christmas decorations.")

computation on a Google Cloud instance (4 virtual cores of the Intel Xeon processors, 32GB of RAM, the algorithm was effectively using only 2 cores at a time and no more than 12GB of memory). The resulting model can be found in `context2vec/syn2015_model_1Mwords_50epochs.zip`.

Since I was not able to train the contex2vec language model on a large corpus which is especially crucial for the Czech language as a fusional one, I could not get good prediction accuracy from the trained Context2vec model. It has not learned to model the Czech language in a sufficient way resulting in bad accuracy on predicting the centre word given its context (see example in Table 5.2), and as a result, it has shown to be unusable for different word sense disambiguation task.

5.3 Using the ELMo model

Unlike Context2vec, ELMo has become popular in the NLP community resulting in multiple pretrained models publicly available on the internet. Since the "official" implementation of the ELMo model created by the authors of the original paper [34] uses ASCII like character encoding, it is not suitable for language using special characters (i.e., Czech). Fortunately,

so-called ELMoForManyLangs implementation of the ELMo model which is using Unicode character encoding has been introduced [44]. Along with this implementation, pre-trained models for many languages (including Czech and German) has been published [45]. In the publicly available ELMoForManyLangs repository is no information about the license of these models – only citation [44] information are provided. I am including these models and the ELMoForManyLangs algorithm in the attached medium since the algorithm need to be configured before use.

The Czech ELMo model has been trained on 20 million words data randomly sampled from the Czech Wikipedia[36] and other publicly accessible internet pages in the Czech language. According to the authors of this model, the training took them about three days using the powerful NVIDIA Tesla P100 GPU. [44]

I have been using the ELMoForManyLangs programmatically via the `Embedder` Python object. This object has allowed me to easily obtain ELMo vector word representations for each word of a particular sentence by simply passing an array of sentences where a sentence is an array of strings (words) to the `sents2elmo()` method of the object. By passing the `output_layer` parameter to the method I have been able to choose in which ELMo layers am I interested in (average of all the three layers, the context-independent character level convolutional network layer, the first layer of the bi-directional LSTM (syntactic dependencies), the second layer of the bi-directional LSTM (semantic meaning) or all the layers at once). The `sents2elmo()` method returns an array of Numpy arrays (sentences) of word embeddings (each embedding consists of 1024 values in this model).

Thanks to the existing model pre-trained on a large Czech textual dataset and the ability to pick different layers of ELMo each representing a slightly different notion of the word, ELMo along with this pre-trained model will be used for the classification and clustering algorithms in this thesis.

5.4 Word sense binary classification

To turn the word sense disambiguation task into a classification one, I have specified the task definition as follows: "For each two pairs of the word usage examples in a sentence where the paired words are always based on the same lemma determine whether they express the same meaning or not." This specification turns the problem into a binary classification. I have been pairing only words based on the same lemma to avoid an enormous confusion which would occur if comparing totally different words (i.e., "house" and "to

[36] `https://cs.wikipedia.org/`

word	sense_id	example
hlavní	3	Hlavní příčinou naší porážky byly chyby v obraně.
hlavní	4	Hlavní město Španělska je Madrid.
růže	5	Dostala jsem tři nádherné rudé růže.
růže	6	Růže, odborně erysipel, se projevuje na kůži (ohrajičené zánětlivé ložisko, jazykovitě se šířící do okolí – pálí, svědí) a celkově (vysoká horečka, zimnice, bolest hlavy, nevolnost).
růže	7	Na tváři ještě měla zbytky růže.
nádraží	8	Vlak vjížděl zvolna do nádraží.
nádraží	9	Jede nám to v pět z autobusového nádraží.
být	10	'Je' Bůh? Je vůbec na světě nějaká spravedlnost?
být	10	Dědeček už tu "'není"'.
být	10	"'Být"', či"'nebýt"'? To je, oč tu běží."

Table 5.3: A preview of the dataset based on the Czech Wiktionary XML dump restricted to adjectives, substantives and verbs.

be") to find if they express the same meaning or not. "Expressing the same meaning" means being in the same `sense_id` class of the dataset (Table 5.1).

5.4.1 Preparation of the classification dataset

I have decided to use only the substantives, adjectives and verbs for the classification dataset, since other categories of words usually do not occur in multiple senses in Czech or are not stand-alone words at all (e.g., prepositions).

After loading the complete Czech adjective, substantive and verbs dataset (based on the Wiktionary XML dump) corresponding to the specification described above (example in Table 5.1) the first step was to remove words with only one example usage in a sentence. Such words would not be useful for this dataset while no pairs containing them could be created. This has resulted in a dataset containing 12021 sentence usage examples of 4099 unique words (on average ~ 2.93 examples per word). See a preview of this dataset in Table 5.3.

In the next step I have filtered out words which were assigned to only one word sense class as well as words which were not assigned to any word sense class containing more than two examples of usage in a sentence. The reason for this filter is to select only words based on which a complex word sense comparison dataset can be created (the dataset must contain at least the following tuples per each word: different words of the same meaning, different words of a different meaning). I have written a script which applies such filter and generates a labelled (1 if the word meanings are the same, 0 otherwise)

word	sentence_a	sentence_b	are_equal
voda	praskla trubka a všude se rozlila voda	nalej mi, prosím, sklenici vody	True
voda	nalej mi, prosím, sklenici vody	plodová voda	False
slovo	dal mu své slovo	měl slovo na úvod	False
slovo	věty se skládají ze slov	to slovo jsem nikdy neslyšel	True

Table 5.4: A preview of the coupled words classification dataset.

dataset containing unique couples of words. The script has also made all the example sentences and words lower-cased. See a preview of the output dataset in Table 5.4.

However, there is an issue with the generated coupled word examples dataset. In most cases it does not say which word in the example sentence is the target one (the word which the example focuses on – i.e., the word *bank* in a couple of sentences "the river and its *banks*" and "I've got money in the *bank*"). It would be useful to have some kind of annotation to highlight the target word. I have chosen to annotate such word by adding "@" before and after it to the sentence (bank → @bank@).

Unfortunately, it is not so simple to find all the occurrences of the target word in a Czech sentence due to its morphological variability. A solution to this issue would be to use a lemmatiser or a stemmer on the whole example sentence, but there are almost no reasonably good ones for Czech. One of the few is a part of the MorphoDiTa tool introduced in the first chapter of this thesis. MorphoDiTa tool offers a very precise lemmatiser, but since it is primarily implemented in C and the Python binding is poorly documented, I have decided to use the online MorphoDiTa tool [37]. I have created a script which has generated all the distinct sentences from the coupled examples dataset created above, and I have manually generated a lemmatised version of these sentences using the MorphoDiTa online tool. Afterwards, I have loaded the lemmatised versions of the sentences back to the Python code and have created a dictionary where keys have been the original sentences and values have been the lemmatised version.

To add annotations around the target words in the example sentences, a simple lookup of the position of the target word in the lemmatised sentence and annotating the word at the same position in the original example sentence has been used. This, of course, presumes that by lemmatising a sentence, the

[37] `http://lindat.mff.cuni.cz/services/morphodita/`

word	sentence_a	sentence_b	are_equal
voda	praskla trubka a všude se rozlila @voda@	nalej mi, prosím, sklenici @vody@	True
voda	nalej mi, prosím, sklenici @vody@	plodová @voda@	False
slovo	dal mu své @slovo@	měl @slovo@ na úvod	False
slovo	věty se skládají ze @slov@	to @slovo@ jsem nikdy neslyšel	True

Table 5.5: A preview of the annotated coupled words classification dataset. The preview is based on Table 5.4.

order of words would not change and no words would disappear. If the lemma of the word was consisting of more than one part (i.e., "rozběhnout se", "sejít se") I have used the longer word as a lookup keyword – and it has shown that for the Czech language this solution works quite well. See a preview of the annotated dataset in Table 5.5.

The last step of creating the binary classification dataset is to compute the word embeddings from the Czech ELMo model for each annotated word given its context (the surrounding sentence). I have used the mentioned ELMoForManyLangs Czech pre-trained model and generated word embeddings by passing the example sentences one by one to the model and then selecting the word embedding vectors at the positions corresponding to the positions of the annotated words from the output of the ELMo model.

Each row of the final dataset consists of the lemmatised word form, a couple of example usages of the word in a sentence (for debugging and analysis purpose), three word embedding vectors (each of size 1024) from the ELMo model per sentence (in total 6 word embedding vectors for two sentences), the sense equality label as well as the cosine similarity and Euclidean distance (feature engineering) of each two corresponding word vectors. This dataset has 6190 examples, however, it is very unbalanced in terms of positive and negative examples. According to the way this dataset has been created, it is logical that negative examples will occur much more frequently than positive ones. Specifically, the dataset contains 4914 negative examples and only 1277 positive ones. I have balanced the dataset in order to make it behave correctly in the binary classification task. The final size of the balanced ELMo Czech dataset is 2554 examples formed by 487 unique words (on average ~ 5.24 examples per word). All the transformations above can be found in the `classification/classification_dataset_preparator.ipynb` Jupyter notebook.

5.4.2 Application of different classifiers

There are multiple ways of using the data from the dataset created in the previous section as an input to a classification algorithm. I have focused on following input data selections:

1. using all the ELMo output vectors ($6 \cdot 1024$ values),
2. using a certain selection of ELMo layer vectors (each layer $2 \cdot 1024$ values),
3. using the feature engineered values (cosine similarity or Euclidean distance) which are of much lower dimension.

Moreover, multiple classifiers can be used as well. In the following sections, individual classifiers with various inputs will be analysed.

5.4.2.1 Train, validation and test dataset split

Since the dataset contains multiple sentence examples per each word and the example sentences can be repeated per each word, a simple train, validation, test split cannot be used. For this reason, I have implemented a custom dataset split function (in `classification/word_sense_classifier.ipynb` Jupyter notebook) which does the split based on the particular words rather than examples. I have applied this function twice (first on the whole dataset, then on the test part) on the dataset to obtain train, validation and test datasets. The final train dataset contains 1653 examples; validation one contains 675 examples and, finally, the test dataset consists of 226 examples.

5.4.2.2 Logistic regression

The simplest classification algorithm used in this thesis is the logistic regression. When used with the cosine similarity of the three ELMo vectors it has not been able to fit the data at all. The same has applied for the Euclidean distance between the ELMo vectors. On the other hand, when the raw ELMo vectors have been passed to Logistic regression, it has over-fitted itself during the training process and failed on the validation dataset. See Table 5.6 for the results.

5.4.2.3 k-nearest neighbors classifier

I have tried to apply the kNN classifier on the dataset, but the results have not been satisfying. It has not been able to fit or predict the data at all. I have been using multiple values of the parameter k varying from 1 up to 30. See Table 5.6 for the results.

	cos. sim.	Eucl. dist.	all ELMo	ELMo 1	ELMo 2
log. r. train	0.53	0.56	0.99	0.98	0.99
log. r. val.	0.47	0.48	0.53	0.52	0.54
kNN train	0.6 (k=19)	0.63 (k=19)	0.76 (k=5)	0.73 (k=6)	0.76 (k=5)
kNN val.	0.55	0.53	0.52	0.57	0.51
Ada. train	0.67	0.77	0.76	0.75	0.74
Ada. val.	0.51	0.52	0.54	0.57	0.53

Table 5.6: Table of accuracies achieved by logistic regression (log. r.), kNN and Adaboost classifiers on the ELMo classification dataset (cosine similarity, Euclidean distance, all the three vectors from ELMo, individual vectors from the ELMo's bi-directional language model).

5.4.2.4 AdaBoost classifier

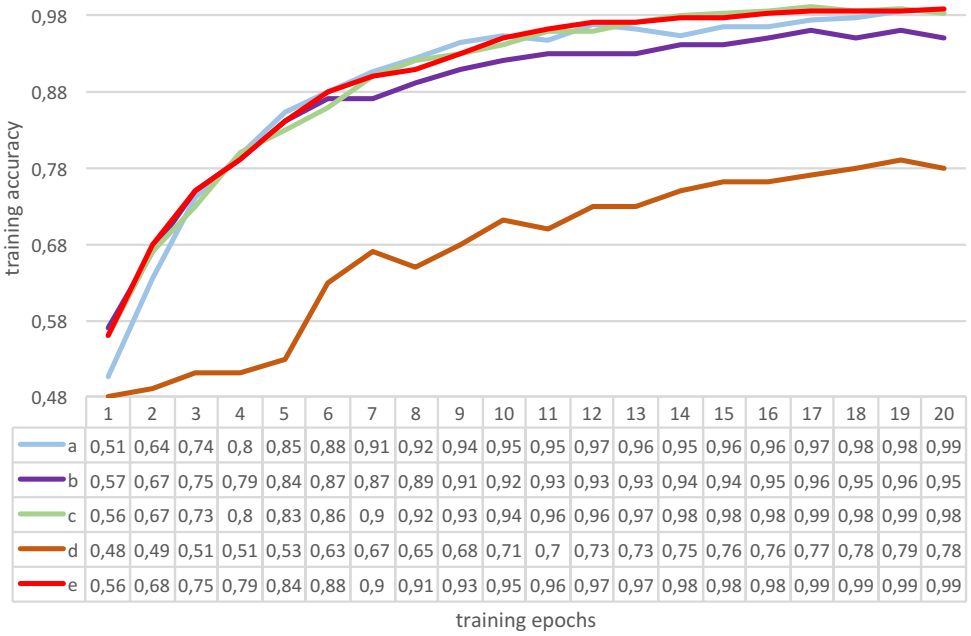
I have been using the ensemble Adaboost classifier with various hyperparameters (number of estimators and learning-rate). However, I have not been able to get a good result with any of the hyperparameter settings. The results presented in Table 5.6 have been achieved by setting the number of estimators to 1000 and learning rate to 1.

5.4.2.5 Deep neural network

Finally, I have applied various modifications of a deep neural network on the classification dataset. The size of the input layer has always corresponded with the size of the input dataset (i.e., 3 for cosine similarity dataset, 6144 for the dataset containing all the ELMo vectors), whereas the last, fully connected layer has always consisted of a single neuron with a sigmoid activation function. However, I have been playing around with the intermediate hidden layers. The binary cross-entropy loss along with Adam optimiser has been used in all experiments.

At first, I have focused on a classification based on all of the ELMo vectors. I have created a deep neural network model containing 3 hidden layers, each made of 1024 (the size of one ELMo vector) neurons with ReLU activation functions and a batch size of 125. This model has been able to fit the data perfectly almost instantly after start leading to a quick over-fitting of the model. The training accuracy of this model architecture after 25 epochs was 0.99 and the validation accuracy was about 0.58. As a result, I have dropped the two hidden state layers, leaving the model with a single layer of 1024 neurons. However, it has still converged very quickly. I have tried another two smaller models, one with just a smaller batch size and the other one

Figure 5.1: A graph visualising the training process of the deep neural network used for word sense binary classification. The architecture of the model **a**: 3 hidden layers each containing 1024 neurons, batch size 125; the architecture of the model **b**: 1 hidden layer containing 1024 neurons, batch size 125; the architecture of the model **c**: 1 hidden layer containing 64 neurons, batch size 30; the architecture of the model **d**: 1 hidden layer containing 3 neurons, batch size 30; the architecture of the model **e**: 1 hidden layer containing 32 neurons, batch size 100.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
a	0,51	0,64	0,74	0,8	0,85	0,88	0,91	0,92	0,94	0,95	0,95	0,97	0,96	0,95	0,96	0,96	0,97	0,98	0,98	0,99
b	0,57	0,67	0,75	0,79	0,84	0,87	0,87	0,89	0,91	0,92	0,93	0,93	0,93	0,94	0,94	0,95	0,96	0,95	0,96	0,95
c	0,56	0,67	0,73	0,8	0,83	0,86	0,9	0,92	0,93	0,94	0,96	0,96	0,97	0,98	0,98	0,98	0,99	0,98	0,99	0,98
d	0,48	0,49	0,51	0,51	0,53	0,63	0,67	0,65	0,68	0,71	0,7	0,73	0,73	0,75	0,76	0,76	0,77	0,78	0,79	0,78
e	0,56	0,68	0,75	0,79	0,84	0,88	0,9	0,91	0,93	0,95	0,96	0,97	0,97	0,98	0,98	0,98	0,99	0,99	0,99	0,99

with much a smaller hidden layer (64 neurons) but they all performed very similarly. The best result I have been able to reach with the neural network on this dataset has been achieved by utilising a single hidden layer containing 32 neurons with the batch size of 100 trained for 20 epochs. The accuracy of this setup has been 0.99 on the training dataset and 0.61 on the validation dataset. See Figure 5.1 for the visualisation of the first 20 epochs of the training process of the mentioned neural network architectures.

When applying the above mentioned neural network architectures to the datasets consisting of particular ELMo layer vectors, the network has behaved very similarly, and the results have also been very close to the ones mentioned in the previous paragraph. Also, the best results have been achieved on the same model architecture resulting in ~ 0.98 accuracy on the training dataset and ~ 0.59 accuracy on the validation dataset. Furthermore, I have tried to use the neural network on the engineered features (cosine similarity and

Euclidean distance of the ELMo vectors) but it was not able to even fit the training data.

5.4.3 Classification conclusion

Overall, I have achieved the best results on the validation dataset by using one hidden layer (32 neurons) neural network described above with the batch size of 100 and by passing all the word vectors from ELMo to the model. This architecture has achieved 0.63 accuracy on the final test dataset. This result is not the supreme one considering the fact that the task is a binary classification.

However, it must be seen that the dataset is very small and the dimensions of the ELMo vectors are relatively large. Moreover, the dataset contains many wrong examples (misspelt words, bad usages of the particular word sense in a sentence). Sometimes it has been hard for me as a Czech native speaker to correctly decide if the two sentences in the dataset correspond to the same word sense or not. Furthermore, the dataset tends to differentiate between small nuances for example based on the speaker ("the weather is nice" (general meaning) and "to predict the weather, we use a sophisticated model" (expert meaning) would be marked as two different word senses).

From the results of the application of different classifiers to the data in previous sections, it is evident that the engineered features (cosine similarity and Euclidean distance) do not carry any useful information about the sense of the word. On the other hand, using all the three vectors (character-level word representation, the first layer of the bi-directional LSTM representation, the second layer of the bi-directional LSTM representation) has shown to be the most accurate way to disambiguate between different word senses. However, using the vectors from the first layer of the bi-directional LSTM has been reasonably successful as well. With slightly worse performance, the vectors from the second layer of the LSTM could also be used for this task. This corresponds to the fact that the character-level word representation from the convolutional neural network does not say almost anything about the particular word sense.

5.5 Word sense clustering

Another way to approach the word sense disambiguation task is by using a clustering algorithm. Fortunately, this approach is unsupervised, so there is no need for hard-to-acquire labelled data (although the labelled data will be used to evaluate the accuracy of the clustering algorithm and to weight the ELMo vectors).

5.5.1 Preparation of the clustering evaluation dataset

The preparation of the evaluation dataset for a clustering algorithm is much easier than for the classification one. Starting with the dataset corresponding to the format shown in Table 5.3, annotations of the target words and their ELMo word embeddings have to be computed in the same way as they were the section 5.4.1. However, in this case, the examples will not be coupled. The word sense class identification column (`sense_id`) will be kept intact so that it can be used for evaluation of the clustering algorithm results. The final dataset consists of three ELMo layer vectors ($3 \cdot 1024$ values) and the (`sense_id`) label. The dataset preparation script can be found in `clustering/clustering_dataset_preparator.ipynb`.

5.5.2 Evaluation of the clustering approach

I have used the k-means clustering algorithm since it has made the most sense for this task. Thanks to the `sense_id` column, I have been able to evaluate the accuracy of the clustering results easily. I have chosen to use the V-measure score [46] which is a combination of the homogeneity and completeness scores for comparing the predicted clusters with the labels. Essentially, the V-measure is a classification accuracy metric which is independent of the absolute values of the labels (the k-means clustering algorithm can name each class in numerous different ways). The following experiments can be found implemented in `clustering/word_sense_clustering.ipynb`.

I have applied the k-means algorithm to the whole dataset with the number of clusters k equal to the actual number of clusters in the dataset (the ground truth) given to it. The results have been very compelling; the accuracy (measured by the V-measure) has reached **0.94** which is an excellent result.

Unfortunately, determining the right number of clusters k based on the data has shown to be a problem. There are a few ways to find a suitable value for k. I have decided to go with the analysis of inertia values and Silhouette scores.

The inertia value is a within-cluster sum of squared distances between individual data points and mean of the particular cluster. This value is not normalised, but the lower the value, the less distant are individual elements of the cluster from each other. If the number of clusters is equal to the number of data points, the inertia value would be zero. [47]

To determine the right number of clusters k using the inertia value, an "elbow" must be found in the graph visualising the inertia values over the number of clusters k. This "elbow" corresponds the fact that before the optimal value k the within-clusters distances were greater than after reaching this value.

Figure 5.2: A plot visualising the inertial values based on the number of clusters of the k-means clustering algorithm. Yellow data point is the optimal ground truth number of k clusters.

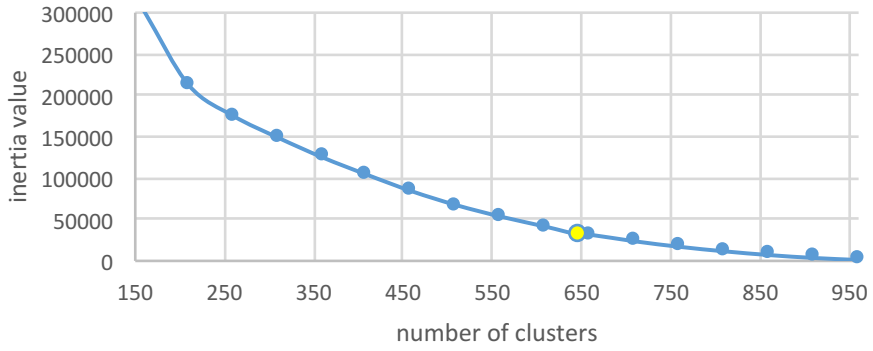
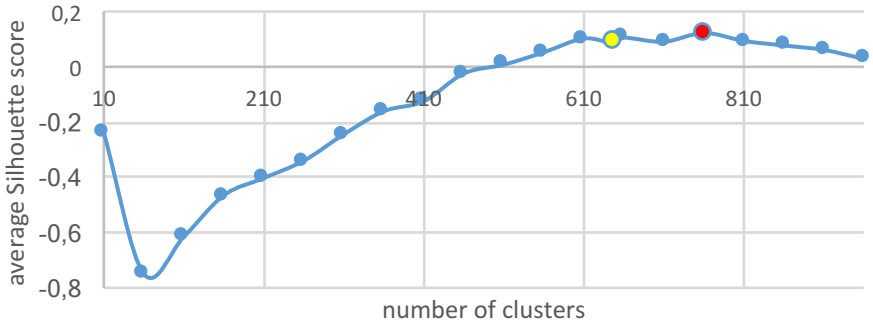


Figure 5.3: A plot visualising the average Silhouette scores (average of Silhouette scores of elements from the dataset) based on the number of clusters of the k-means clustering algorithm. Yellow data point is the optimal ground truth number of k clusters. Red data point (best one) is the the point with the highest average Silhouette score.



Due to the high computational complexity of the calculation of the k-means clusters on such high-dimensional dataset, I have been using a sample of 1 000 examples from the original dataset (647 sense classes) to obtain the inertia values for the different number of clusters k. As it can be seen from the plotted inertial values in Figure 5.2, there is no noticeable "elbow" in the plot. This might be due to the fact that the data are high-dimensional, noisy and almost evenly distributed in the vector space. Using the inertial value has not shown any useful results in terms of determining the number of k clusters.

The other approach uses so-called Silhouette scores [48]. The Silhouette score basically shows how similar is the element to its own cluster in a comparison with all the other clusters. The value ranges from -1 to 1. The higher the

value the better is the element matched to its own cluster (it is similar to its own cluster and dissimilar to neighbouring clusters).

I have used, again due to computational complexity, a sample of 1 000 examples (647 sense classes) from the dataset to determine number of k clusters by the Silhouette score. The results are visualised in Figure 5.3. I have achieved better results in obtaining the number of k clusters by the Silhouette scores than by the inertial values. However, it can be seen from the Silhouette average scores plot that in none of the clusters are the examples well matched to their cluster. This shows that the input data are vague and noisy.

5.5.3 Weighting ELMo vectors for clustering

Until now, I have been using all the three ELMo vectors equally for the clustering algorithm. As a reminder, the first vector m_0 corresponds to character level information, the second vector m_1 is the hidden state of the first layer of LSTM which matches to syntactic information and the third vector m_2 is the hidden state of the second's layer of LSTM which corresponds to semantic and syntactic information. All the following experiments will be using the true number of clusters in the dataset.

The paper which has introduced ELMo [34] suggests using a linear combination of the three vectors

$$\text{ELMo}_{\text{embedding}} = s_0 m_0 + s_1 m_1 + s_2 m_2,$$

where s_0, s_1, s_2 are softmax-normalised weights and m_0, m_1, m_2 are ELMo vectors. Applying a linear combination of the original vectors would end up in lower dimensional input (one vector instead of three) to the k-means, highlighting the most important features and possibly better results.

Since the clustering results on the not weighted ELMo vectors dataset has been promising so far, I have decided to just randomly try a vast amount of possible weights s_0, s_1, s_2 to get the notion of which layers of the ELMo language model are essential for the word disambiguation task. First of all, I have split the dataset into a train and a test one. Then I have been generating random values (uniformly distributed) in the range from -1 to 1 of s_0, s_1, s_2 for 100 iterations. After each triple of weights s_0, s_1, s_2 has got generated, I have evaluated the training accuracy of the ELMo dataset weighted by these weights.

The best result, I have been able to achieve with accuracy 0.96, has been obtained by applying weights $s_0 = 0.69$, $s_1 = 0.082$, $s_2 = 0.26$ on the dataset. This means that the character level information has been the most

word	sense_id	example
Karotte	50882	Karotten putzen
Donner	21752	ein ferner D.
Ausbringung	78758	Ausbringung der Aerosole durch Hubschrauber
Ausbringung	78758	die Ausbringung von Klärschlamm verbieten
Ausbringung	78758	die Ausbringung von Düngemitteln
Inspiration	50328	Inspiration holt sich der italienische Regisseur stets aus dem eigenen Leben.
Inspiration	50329	Bei der Inspiration ohne Gerät braucht er Unterdruck in der Lunge.
Inspiration	50328	künstlerische, dichterische, musikalische Inspirationen
Inspiration	50328	die Inspiration eines Erfinders, eines Dichters

Table 5.7: Randomly chosen examples from the German dataset [49]. `sense_id` is a word sense class identification.

important (probably for determining whether the words are of the same lemma or not), the syntactic information has not helped a lot in the word sense disambiguation task and the semantic information (which is important for determining individual word senses) has been reasonably taken into account. The test accuracy of this setup has been **0.97**.

5.6 Application of the proposed methods on the German language

German is somewhere between Czech and English in terms of the complexity of its morphology. Thus the application of the previously proposed methods on the German language could result in slightly better overall accuracy. The supervisor of this thesis provided me with an additional German language word sense disambiguation dataset [49] containing substantives and adjectives (see an example from this dataset in Table 5.7. This dataset contains word lemmas, their examples of usage in a sentence and their word sense class' ids. In this section, I will apply the methods which have already been applied to the Czech language above on the German language as well.

Unfortunately, due to copyright reasons, I cannot include the German dataset in the attached medium. As a consequence, the intermediate generated datasets which contain words, and their usage examples will not be included, too. However, the clustering and classification datasets containing only ELMo word embeddings will be included in the attached medium (it is impossible or at lest very hard to obtain the original sentence example based on ELMo word embeddings of the target word).

5.6.1 Preparation of the German dataset

The German dataset [49] I have been working with is much bigger (120 060 examples, 38 258 processable examples with at lest one sense containing at least two example sentences) and more accurate in comparison with the Czech dataset I have obtained from Wiktionary. However, there are still some issues with this dataset. First of all, the examples sometimes contain abbreviations of the target words. For example for the target word "Jungtier" there is an example sentence "die Aufzucht der J-e" where "J-e" stands for "Jungtiere". Fortunately, these abbreviations can be easily found and turned back into full word representation.

The second minor problem about this dataset is that is might use general word ideas instead of explicitly expressing them (for example "He has bought the obj." where "obj" stands for any object). These sentences will not be used for the creation of the ELMo embeddings dataset.

Other than that, the dataset is well structured and does not contain many misspelt words or incorrect examples. However, in my opinion, it tends to recognise too much of a word meaning details which results in placing two words which would an average person say are of the same meaning into two different word sense classes – which has been a problem with the Czech Wiktionary dataset as well.

The following operations over the original German dataset can be found implemented in `german/german_elmo_dataset_preparator.ipynb` Jupyter notebook.

In the first step, I have replaced the abbreviations in example sentences with the original word forms. The original German dataset does not contain word annotations which means that I am not able to determine which of the words in the example sentence corresponds to the target word. Since German is not morphologically as rich as Czech, I have decided to find words based on the particular lemma by simply finding words which start with the lemma using a regular expression. This method has proved to work for most of the words in the dataset.

I will not be going into much detail with the further transformations which have been applied to this dataset since they have been almost the same as the ones for the Czech language explained in previous sections. I have used the German pretrained ELMo model [44] to create both classification and clustering ELMo word embedding datasets.

5.6.2 German word sense binary classification

In this section, I will evaluate the results of the binary word sense classification applied to the German dataset. The algorithms and architecture of the models will be the same as the ones used for the Czech language in Section 5.4.2.

The cosine similarity of the corresponding ELMo vectors has not, again, shown to be carrying any important information in terms of word sense disambiguation task. Logistic regression and Adaboost classifier were not even able to fit the training data resulting in train accuracy $\sim$ 0.55 and validation accuracy slightly lower. The same goes for the neural network classifier (one hidden layer of 64 neurons with ReLU activation function and single neuron output layer with sigmoid activation function) which achieved only 0.56 training accuracy (testing accuracy 0.53). Practically the same results have been obtained while using the Euclidean distances between corresponding ELMo vectors.

I have been focusing mostly on the neural network classifier using the original three ELMo vectors as its input, while this combination has led to the most promising results in the case of the Czech language dataset. Since the German dataset is much larger than the Czech one, the neural network has not converged that quickly. As a consequence, I was capable of taking advantage of multiple hidden layers of the neural network and a higher number of neurons in each of them.

Unfortunately, I have not been able to achieve higher validation accuracy than $\sim$ 0.6 (using the ELMo vectors from the second layer of the language model). Even if the training process curve looked promising, in the end, the model has not been able to differentiate between word senses with reasonable accuracy. All the neural network based classifier has been able to fit the data (from 0.85 to 0.97 training accuracy), but the result accuracy on the validation dataset has not been better than 0.6. The best results have been achieved by utilising a double hidden layer neural network with 512 neurons (ReLU activation function) in each hidden layer and a single neuron output layer with the sigmoid activation function. The classification algorithm for the German dataset can be found in `german/german_word_sense_classifier.ipynb`.

5.6.3 German word sense clustering

The clustering ELMo dataset contains 38 258 in-sentence usage examples of 15 608 distinct word senses. When taking into account also the high dimension of the three ELMo embedding vectors ($3 \cdot 1024$ values), it is evident that computing k-means clusters with $k = 15\ 608$ (the ground truth k) will be computationally very expensive. The following experiments can be found implemented in the `german/word_sense_clustering.ipynb`

Jupyter notebook. Accuracy of the following results has been calculated using V-measure.

I have first applied the k-means algorithm to the whole dataset providing it with k equal to the actual number of distinct word sense classes in the dataset. This has resulted in overall 0.952 accuracy which might seem like a worse result than on the Czech dataset, however if we look closer at the data, we find out that in this dataset there are often many examples per word sense (in the Czech dataset, there has been usually one or two examples per word sense). The Czech clustering dataset has contained on average 1.43 examples per word sense; the German one contains on average 2.45 examples per word sense and the clustering accuracy has been roughly the same. This means that this time the clustering algorithm was more capable of accurate disambiguation between different word senses.

However, the problem with determining the number k of clusters to be found using the k-means algorithm stays the same even with the German dataset. For the following examples, I have used a much smaller sample from the dataset containing 1 000 word sense examples of 385 individual word senses. Using the Silhouette score has not shown any good results and performed worse than on the Czech dataset (almost for every k except the highest ones the average Silhouette score was a negative number). There has been no "elbow" visible in the plotted inertia values as well. Overall, the clustering algorithm has shown decent results on the German dataset. However, I have not been able to find a way to determine the desired number of clusters to be found accurately.

5.6.4 Weighting ELMo vectors using the German dataset

As it has been already stated in the Section 5.5.3, a linear combination of the three original ELMo word embedding vectors should be used in the downstream task. As a remainder, the linear combination of the original ELMo word embedding vectors is calculated as

$$\text{ELMo}_{\text{embedding}} = s_0 m_0 + s_1 m_1 + s_2 m_2,$$

where s_0, s_1, s_2 are softmax-normalised weights and m_0, m_1, m_2 are ELMo vectors.

Due to the size of the dataset, I have been forced to do the following experiments on the smaller dataset sample (1 000 examples, 385 individual word senses) from the previous section. On the sampled dataset I have achieved the highest clustering accuracy (measured by V-measure) of 0.91 by placing $s_0 = 0.79$, $s_1 = 0.02$ and $s_2 = 0.1$. This corresponds to the fact that in the clustering task the algorithm first needs to differentiate between

individual word lemmas and their possible morphological variations and after that, it can focus on the semantic information to disambiguate between individual word senses.

Chapter 6

Conclusion

The goal of this thesis was to get familiar with the linguistic methods used in language modelling with the focus on languages with rich morphology and survey the state-of-the-art machine learning algorithms for natural language processing. Finally, I was to apply the state-of-the-art methods on the Czech and German language word sense disambiguation task and evaluate the results.

In the first part of this thesis, I have made an introduction to the field of linguistics and have explained some of the most relevant concepts in this area. Later, I have used these concepts to describe traditional as well as modern approaches to natural language processing.

Before I have dived into the state-of-the-art language processing methods, I have made an in-depth survey of the currently used machine learning algorithms which are utilised in the modern language processing models. At the end of the theoretical part of this thesis, I have described the modern natural language processing methods focused on the word sense disambiguation capabilities.

Lastly, I have acquired the necessary Czech and German word sense disambiguation datasets and have applied the most advanced natural language processing model (ELMo) up to date in terms of the capability of modelling polysemy on them. Afterwards, I have used the supervised classification along with the unsupervised clustering algorithms to disambiguate between different word senses.

The word sense disambiguation task is extremely hard to be accurately approached by machines. The Czech language as a fusional one is very dynamic and needs to be processed by algorithms which can work with

morphological analysis in some way. Even though the German language does not have that rich morphology, it certainly is not an analytical language with no morphological derivations as well.

An issue I have come across was the lack of a reliable dataset of a decent size for the Czech language. Moreover, the Czech dataset obtained from Wiktionary contains a lot of misspelt words and incorrect word sense examples. For the German language, I have been provided with a lot better dataset containing sufficient amount of examples needed for the word sense disambiguation task by the supervisor of this thesis.

Unfortunately, both these datasets are sometimes going into much detailed classification in terms of "intuitive" word sense disambiguation (by "intuitive" I mean that the algorithm should be able to understand meanings of words on the human speaker level, not on the academic level). For example the word "alkohol" ("alcohol") referrers to different word senses in sentences "Alkohol podávaný v malých dávkách neškodí v jakémkoliv množství." (alcohol in colloquial language, translated: "Small doses of alcohol do not harm in any amount.") and "Na stěně visel zákaz prodeje alkoholu mladistvým a lidem v podnapilém stavu." (beverage, translated: "There was a ban on selling alcohol to juveniles and people in a drunken state.") according to the Czech dataset. These two different word sense examples would be, in my opinion, classified as a single word sense by a native speaker.

Similarly, the same issue occurs in the German dataset as well. It is not that frequent as in the Czech dataset, but by randomly going through it, I was able to find such an example. For example, the word "Metronom" ("metronome") is classified into two separate word sense classes in examples "Das Metronom kann man schneller und langsamer einstellen." ("The metronome can be set faster and slower.") and "Das Metronom gibt ein gleichmäßiges Tempo durch gleichmäßiges Anschlagen von Notenwerten vor." ("The metronome provides a steady tempo by evenly striking note values."). I think that in this case, the authors of the dictionary were thinking of a metronome in two ways – a general metronome (the first sentence) and a metronome used by musicians (the second sentence). However, based solely on the example, I think that a native speaker would classify these as belonging to the same word sense class.

In my opinion, the word sense disambiguation task would be useful if it could understand word senses in the same way as an average human native speaker would. The detailed level of differentiating between distinct word senses presented in formal dictionaries is not that useful for everyday use (i.e., controlling computer by having a natural voice conversation with it). Therefore I think, that for the supervised approach to word sense

disambiguation and evaluation of unsupervised algorithms used to attack this task a good dataset based on everyday language must be created.

First of all, I have tried to approach the word sense disambiguation task by a binary classification. The general idea behind using classification methods was that usually these perform better (due to their discrete characteristics) than other methods on various tasks. The only reasonable way to fit the word disambiguation task into a classification one I have been able to find has been comparing each two word sense examples and determining whether they are of the same meaning or not.

To avoid a massive amount of couples of words which are not even based on the same lemma (and thus in most cases, they are of different meanings) I have been comparing only words derived from the same lemma. However, the resulting dataset was unbalanced, and I have been forced to get rid of many negative (examples are not of the same meaning) examples.

The results of the binary classification approach to the word sense disambiguation task have not been auspicious so far. The best binary classification accuracy 0.63 has been achieved on the Czech dataset. In my opinion, the biggest issue with this approach is that it is very prone to making many mistakes if the input dataset is not perfect. Moreover, word senses can be similar or dissimilar with some distance from each other, and this distance may vary on the particular word – such thing might be challenging to be correctly approached using a binary classification algorithm.

The second approach I have tried has been utilising an unsupervised clustering algorithm k-means, and the results have been evaluated compared to the labelled dataset by using V-measure. Overall, the clustering approach has shown to be much more accurate than the binary classification one resulting in the best accuracy 0.97 on the whole dataset (using the ground truth number of cluster obtained from the dataset), measured by V-measure. However, I have not been able to successfully determine a suitable number of clusters to be used in the k-means algorithm.

Although the results of the application of the state-of-the-art language processing models on the Czech language dataset have not been that positive, I have realised the problems causing the lower accuracy behind the results. However, a solution to these problems would require a manual creation of a massive appropriate dataset and training individually different parts of the state-of-the-art natural language processing models which would be computationally very expensive.

From my personal point of view, I have deeply understood a series of advanced machine learning algorithms (varying from convolutional neural networks up

to encoder-decoder alike complex models) from the field of natural language processing as well as understood the basic concepts of linguistics. This work has motivated me in further research in the field of study.

Bibliography

[1] Manning, C. Natural Language Processing with Deep Learning, Lecture 1. 2017, [online], [accessed 2019-03-31], Stanford University. Available from: https://www.youtube.com/watch?v=OQQ-W_63UgQ&t=0s&list=PLqdrfNEc5QnuV9RwUAhoJcoQvu4Q46Lja&index=2

[2] Manker, J. Ling 100 – Introduction to Linguistic Science, Morphological Typology. 26 Feb 2016, [online], [accessed 2019-04-01], University of California, Berkeley. Available from: http://linguistics.berkeley.edu/~jtmanker/Morphological%20Typology%20-%20Spring%202016%20-%20Ling%20100%20Guest%20Lecture.pdf

[3] Zeman, D. Computational Morphology and Syntax of Natural Languages. 08 Nov 2010, [online], [accessed 2019-04-01], Institute of Formal and Applied Linguistics (ÚFAL) at the Computer Science School, Faculty of Mathematics and Physics, Charles University, Czech Republic. Available from: http://ufal.mff.cuni.cz/~zeman/vyuka/morfosynt/msa01-introduction.pdf

[4] Hancox, P. J. SEM1A5, Natural Language Processing. 26 Jan 1996, [online], [accessed 2019-04-01], University of Birmingham. Available from: https://www.cs.bham.ac.uk/~pjh/sem1a5/pt2/pt2_intro_morphology.html

[5] Marcus, M.; Kim, G.; et al. The Penn Treebank: Annotating Predicate Argument Structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, Stroudsburg, PA, USA: Association for Computational Linguistics, 1994, ISBN 1-55860-357-3, pp. 114–119, doi:10.3115/1075812.1075835. Available from: https://doi.org/10.3115/1075812.1075835

[6] Andrew McCallum, C. M. Introduction to Natural Language Processing, Lecture 5: Context Free Grammars. 2007, [online], [accessed 2019-04-02], University of Massachusetts Amherst. Available from: https://people.cs.umass.edu/~mccallum/courses/inlp2007/lect5-cfg.pdf

[7] Manning, C. D.; Raghavan, P.; et al. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008, ISBN 0521865719, 9780521865715.

[8] Neubig, G. NLP Programming Tutorial 1 – Unigram Language Models. [online], [accessed 2019-04-04], Nara Institute of Science and Technology (NAIST). Available from: https://www.youtube.com/watch?v=ERibwqs9p38

[9] Pustejovsky, J. Introduction to N-gram Models, COSI 114 – Computational Linguistics. 2015, [online], [accessed 2019-04-04], Brandeis University. Available from: http://spring2015.cs-114.org/wp-content/uploads/2016/01/NgramModels.pdf

[10] Dan Jurafsky, J. H. M. Speech and Language Processing (3rd ed. draft), Language Modeling with N-Grams. 2018, [online], [accessed 2019-04-04], Stanford University. Available from: https://web.stanford.edu/~jurafsky/slp3/slides/LM_4.pdf

[11] Manning, C. Natural Language Processing with Deep Learning, Lecture 2 | Word Vector Representations: word2vec. 2017, [online], [accessed 2019-04-03], Stanford University. Available from: https://www.youtube.com/watch?v=ERibwqs9p38

[12] Wikipedia contributors. Czech language – Wikipedia, The Free Encyclopedia. 2019, [online], [accessed 4-April-2019]. Available from: https://en.wikipedia.org/w/index.php?title=Czech_language&oldid=890788875

[13] Czech National Corpus. [corpus], [online], [accessed 2019-05-10]. Available from: https://www.korpus.cz/

[14] Straková, J.; Straka, M.; et al. Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Baltimore, Maryland: Association for Computational Linguistics, June 2014, pp. 13–18, [online], [accessed 4-April-2019]. Available from: http://www.aclweb.org/anthology/P/P14/P14-5003.pdf

[15] Knuth, D. E. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997, ISBN 0-201-89684-2.

[16] Spoustová, D.; Hajic, J.; et al. Semi-Supervised Training for the Averaged Perceptron POS Tagger. In *EACL 2009, 12th Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference, Athens, Greece, March 30 - April 3, 2009*, 2009, pp. 763–771. Available from: http://www.aclweb.org/anthology/E09-1087

[17] Rumelhart, H. G. E., David E; Williams, R. J. *Learning internal representations by error propagation*. DTIC Document, 1985.

[18] Fei-Fei Li, S. Y., Justin Johnson. *Lecture 10: Recurrent Neural Networks*. Stanford University School of Engineering, 2017, [online], [accessed 2019-02-27]. Available from: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

[19] Vinyals, O.; Toshev, A.; et al. Show and Tell: A Neural Image Caption Generator. *CoRR*, volume abs/1411.4555, 2014, 1411.4555. Available from: http://arxiv.org/abs/1411.4555

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

[21] Bengio, Y.; Simard, P.; et al. Learning Long-term Dependencies with Gradient Descent is Difficult. *Trans. Neur. Netw.*, volume 5, no. 2, Mar. 1994: pp. 157–166, ISSN 1045-9227, doi:10.1109/72.279181. Available from: http://dx.doi.org/10.1109/72.279181

[22] How the brain decides what to learn. *Stanford University, ScienceDaily*, Oct. 2018, [online], [accessed 2019-03-10]. Available from: www.sciencedaily.com/releases/2018/10/181025142023.htm

[23] Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.*, volume 9, no. 8, Nov. 1997: pp. 1735–1780, ISSN 0899-7667, doi:10.1162/neco.1997.9.8.1735. Available from: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[24] Olah, C. Understanding LSTM Networks. *colah's blog*, Aug. 2015, [online], [accessed 2019-03-15]. Available from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[25] Krizhevsky, A.; Sutskever, I.; et al. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, edited by F. Pereira; C. J. C. Burges; L. Bottou; K. Q. Weinberger, Curran Associates, Inc., 2012, pp. 1097–1105. Available from: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[26] Pascanu, R.; Gülçehre, Ç.; et al. How to Construct Deep Recurrent Neural Networks. *CoRR*, volume abs/1312.6026, 2013, 1312.6026. Available from: http://arxiv.org/abs/1312.6026

[27] Józefowicz, R.; Vinyals, O.; et al. Exploring the Limits of Language Modeling. *CoRR*, volume abs/1602.02410, 2016, 1602.02410. Available from: http://arxiv.org/abs/1602.02410

[28] Li, F.-F.; Johnson, J.; et al. CS231n: Convolutional Neural Networks for Visual Recognition. 2017, [online], [accessed 2019-03-29]. Available from: https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv

[29] Bengio, Y.; LeCun, Y. (editors). *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. Available from: https://iclr.cc/archive/www/doku.php%3Fid=iclr2015:accepted-main.html

[30] Pascanu, R.; Mikolov, T.; et al. Understanding the exploding gradient problem. *CoRR*, volume abs/1211.5063, 2012, 1211.5063. Available from: http://arxiv.org/abs/1211.5063

[31] Srivastava, R. K.; Greff, K.; et al. Highway Networks. *CoRR*, volume abs/1505.00387, 2015, 1505.00387. Available from: http://arxiv.org/abs/1505.00387

[32] Kim, Y.; Jernite, Y.; et al. Character-Aware Neural Language Models. *CoRR*, volume abs/1508.06615, 2015, 1508.06615. Available from: http://arxiv.org/abs/1508.06615

[33] Mnih, A.; Hinton, G. E. A Scalable Hierarchical Distributed Language Model. In *Advances in Neural Information Processing Systems 21*, edited by D. Koller; D. Schuurmans; Y. Bengio; L. Bottou, Curran Associates, Inc., 2009, pp. 1081–1088. Available from: http://papers.nips.cc/paper/3583-a-scalable-hierarchical-distributed-language-model.pdf

[34] Peters, M. E.; Neumann, M.; et al. Deep contextualized word representations. *CoRR*, volume abs/1802.05365, 2018, 1802.05365. Available from: http://arxiv.org/abs/1802.05365

[35] Mikolov, T.; Sutskever, I.; et al. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26*, edited by C. J. C. Burges; L. Bottou; M. Welling; Z. Ghahramani; K. Q. Weinberger, Curran Associates, Inc., 2013, pp. 3111–3119. Available from: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

[36] Melamud, O.; Goldberger, J.; et al. context2vec: Learning generic context embedding with bidirectional lstm. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, 2016, pp. 51–61.

[37] Křen, M.; Cvrček, V.; et al. SYN2015: representative corpus of written Czech. 2015, LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. Available from: `http://hdl.handle.net/11234/1-1593`

[38] Internetová jazyková příručka. [dictionary], [online], [accessed 2019-04-27]. Available from: `http://prirucka.ujc.cas.cz/`

[39] Slovník spisovného jazyka českého. [dictionary], [online], [accessed 2019-04-27]. Available from: `https://ssjc.ujc.cas.cz/`

[40] Lopatková, M.; Kettnerová, V.; et al. *Valenční slovník českých sloves VALLEX*. Praha: Karolinum, 2016, ISBN 978-80-246-3542-2.

[41] Příruční slovník jazyka českého (1935–1957). [dictionary], [online], [accessed 2019-04-27]. Available from: `https://psjc.ujc.cas.cz/`

[42] Wiktionary, the free dictionary. [dictionary], [online], [accessed 2019-04-27]. Available from: `https://www.wiktionary.org/`

[43] Oren Melamud, S. B., Frankie Robertson. context2vec. `https://github.com/orenmel/context2vec`, [online], [accessed 2019-04-18].

[44] Che, W.; Liu, Y.; et al. Towards Better UD Parsing: Deep Contextualized Word Embeddings, Ensemble, and Treebank Concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, Brussels, Belgium: Association for Computational Linguistics, October 2018, pp. 55–64. Available from: `http://www.aclweb.org/anthology/K18-2005`

[45] Fares, M.; Kutuzov, A.; et al. Word vectors, reuse, and replicability: Towards a community repository of large-text resources. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, Gothenburg, Sweden: Association for Computational Linguistics, May 2017, pp. 271–276. Available from: `http://www.aclweb.org/anthology/W17-0237`

[46] Rosenberg, A.; Hirschberg, J. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, 2007.

[47] Clustering. [manual], [online], [accessed 2019-04-27]. Available from: https://scikit-learn.org/stable/modules/clustering.html

[48] Silhouette score. [manual], [online], [accessed 2019-04-27]. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

[49] VACHKOVÁ, M. *Das große akademische Wörterbuch Deutsch–Tschechisch : Ein erster Werkstattbericht. 1 vyd.* Frankfurt am Main: Peter Lang, 2011, ISBN 978–3–631–60567–7.

Appendix A

Acronyms

NN Neural network

CNN Convolutional neural network

RNN Recurrent neural network

LSTM network Long short-term memory network

biLSTM network bi-directional Long short-term memory network

LM Language model

Appendix B

Contents of enclosed DVD

- `readme.txt` ..... the file with DVD contents description
- `classification` ..... the directory of classification methods
  - `datasets` ..... the directory of cached temporary datasets
- `clustering` ..... the directory of clustering methods
  - `datasets` ..... the directory of cached temporary datasets
- `context2vec` ..... the directory of Context2vec model
- `elmo` ..... the directory of ELMo models
- `german` .... the directory of German classification and clustering methods
  - `datasets` ..... the directory of cached temporary datasets
- `syn2015` ..... the directory of SYN2015 corpus and its parser
- `wiktionary` ..... the directory of Wiktionary dump and its parser
- `requirements.txt` ..... the Python 3 environment requirements list
- `thesis.pdf` ..... the thesis text in PDF format