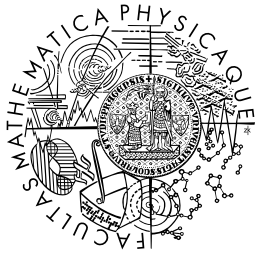

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

**BACHELOR THESIS**

Vladan Glončák

# Artificial Neural Network for Opinion Target Identification in Czech

Institute of Formal and Applied Linguistics

| | |
|---|---|
| Supervisor of the bachelor thesis: | doc. RNDr. Vladislav Kuboň, Ph.D. |
| Study programme: | Computer Science |
| Study branch: | IOI |

Prague 2016

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

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

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

Title: Artificial Neural Network for Opinion Target Identification in Czech

Author: Vladan Glončák

Institute: Institute of Formal and Applied Linguistics

Supervisor: doc. RNDr. Vladislav Kuboň, Ph.D., Institute of Formal and Applied Linguistics

Abstract: The main focus of this thesis is to use neural networks, specifically long short-term memory cells, for identifying opinion targets in Czech data. The side product is a new version of dataset for opinion target identification. For a comparison, previously obtained results for another languages and by employing probabilistic methods instead were listed. The experiment was successful, achieved results are above trivial baseline models and comparable with the results achieved previously.

Keywords: neural network, opinion target identification, sentiment analysis

# Acknowledgment

During the work on this thesis I greatly benefited from the support of many people, some of whom I would sincerely like to thank here.

Firstly I would like to thank to the supervisor of the thesis, doc. RNDr. Vladislav Kuboň Ph.D. Without his great support this thesis would not exist.

Furthermore, I would like to express my sincere gratitude to the consultant of this thesis Mgr. Kateřina Veselovská Ph.D. for her support, patience with responding to long emails and, last but not least, her previous research conducted together with Mgr. Aleš Tamchyna.

Also, I would like to thank to my parents and all my friends for their strong emotional support, especially to Jakub Dargaj, Michael Higgs and my fellow scientist, Ľubomír Ohman.

# Contents

**Introduction** **3**

**1 Sentiment Analysis** **4**
1.1 Obstacles to Opinion Mining . . . . . 4
1.2 Problem Formulation . . . . . 5
1.3 Aspect-Based Sentiment Analysis . . . . . 5
1.4 Machine Learning Methods . . . . . 6
1.4.1 Cross-Validation . . . . . 7
1.4.2 Conditional Random Fields . . . . . 7
1.4.3 Logistic Regression . . . . . 8

**2 Neural Networks** **10**
2.1 Model of a Neuron . . . . . 10
2.1.1 Squashing Function . . . . . 10
2.2 Network Architectures . . . . . 11
2.2.1 Long Short-Term Memory . . . . . 13
2.3 Training . . . . . 15
2.3.1 Back-Propagation Algorithm . . . . . 15
2.3.2 Back-Propagation Through Time . . . . . 17

**3 Word Embeddings and Continuous Bag-of-Words Model** **18**
3.1 Word Embedding . . . . . 18
3.2 Continuous Bag-of-Words Model . . . . . 18
3.2.1 Training . . . . . 21

**4 Dataset** **22**
4.1 Wikipedia Dump . . . . . 22
4.2 Czech Dataset . . . . . 23
4.2.1 Dataset Preprocessing . . . . . 23

**5 Experiment** **25**
5.1 Previous Work . . . . . 25
5.1.1 Experiment with the Czech Dataset . . . . . 25
5.1.2 Recurrent Neural Network for Sentence Classification . . . 26
5.2 Experiment with the Original Model . . . . . 27
5.2.1 Word Embeddings . . . . . 27
5.2.2 Baseline . . . . . 29
5.2.3 Results of the Original Model . . . . . 29
5.3 Modified Model . . . . . 29
5.3.1 Decision Threshold . . . . . 30
5.3.2 Reducing the Number of Models . . . . . 31
5.3.3 Max-Pooling . . . . . 32
5.3.4 Tuning the Number of Long Short-Term Memories . . . . 32

**Conclusion** **34**

**Bibliography** **36**

**List of Figures** **38**

**List of Tables** **39**

**List of Abbreviations** **40**

**Attachments** **41**

# Introduction

Sentiment analysis, or opinion mining, aims to identify and extract subjective information, such as the attitude of the author, in source material. Attitudes, opinions and emotions are all subjective sentiments. Text analysis, natural language processing methods and even machine learning methods are used for this purpose. Whilst they are not facts, sentiments may be helpful in business intelligence, sociology, psychology and related fields.

In general, the most typical task of sentiment analysis is to determine whether a short text or a sentence has positive, negative or neutral polarity. Another common task is to classify a given piece of text as subjective or objective. However, sometimes more detailed-oriented approaches are necessary.

A typical example is user reviews, which usually have several aspects that are mentioned in the review. To obtain useful data from restaurant reviews, it would be important to know the sentiment about food quality, friendliness of the staff and other features, not only the polarity of the whole review.

*Aspect-based sentiment analysis* (ABSA) refers to determining the sentiments on features or aspect of entities. In the example of restaurant reviews, food is an entity, while price and quality are aspects of the entity. Both entities and aspects, to be evaluated later, must be determined first. This roughly divides ABSA into two sub-tasks; the *identification of aspects* and *polarity classification*. The purpose of the experiment introduced in this thesis is the identification of aspects. A fine-grained approach and the consideration of the internal sentence structure is required to carry out those sub-tasks.

Most sentiment prediction systems work by analyzing words in isolation. This approach is simple and comprehensive; however, often the most important information hidden in the context is lost. To pursue aspect-based analysis without requiring a more language-specific approach and engineering, we introduce neural networks.

Neural networks are a family of machine learning models loosely based on central nervous systems of higher animals. Neural networks possibly have a greater ability of generalization than probabilistic models.

Neural networks have been used in sentiment analysis before. In this work we use recurrent neural networks, namely *long short term memory* (LSTM), because syntactic relationships and long-distance dependencies, which we assume are a great influence, may be better modeled with a recurrent neural network rather than convolutional neural network.

After introducing the issue in the first few chapters we perform the main purpose of this thesis: identifying the opinion target, the aspect and entity, that are being reviewed. We implement the previously used model in Python and measure its performance on Czech data. The results are then compared to the results already achieved by the use of probabilistic models and neural networks. We also try to modify the model in order to achieve higher performance.

For the purpose of our experiment, we modify the existing dataset for opinion target identification in Czech, the same dataset that was used for the probabilistic models. Language independence of the approach allows us to compare the results with results from analysis of other languages.

# 1. Sentiment Analysis

There are lots of facts that we remember throughout our lifetime. However, not everything that we say is a fact. Most of what people say are just opinions. Despite that, opinions have always been an important factor when making decisions.

Even before the expansion of the Internet, people would ask their friends for their opinions of various kinds of services. The increasing use of the Internet has also influenced this part of our lives. Instead of asking friends, many people use the Internet to get reviews of movies, restaurants, products they consider buying and so on.

The web is filled with data containing opinions and sentiments. The amount of the data is massive. Obtaining useful information from this enormous amount of subjective sets of data demands new technologies. Major companies and organizations are coming to realize that platforms, such as blogs and discussion forums, are a great influence of other people's opinions, underscoring the need for a technology to elucidate the actual moods. Business and government intelligence are just examples of fields that could make use of such technologies.

There are many challenges related to the analysis of such data. For example, determining which parts of text contain the subjective material. In some interfaces, like Amazon, it is quite easy, but in blog entries it may be more difficult. The main task is identifying the overall polarity of the material. The subjective material can be quite complex, expressing different opinions on different aspects of the topic. Therefore another task is to identify those aspects that are being evaluated.

All those tasks are research areas of *sentiment analysis*. Sentiment analysis has been around for quite a while. In recent years it has enjoyed greater awareness, mostly due to the increasing use of machine learning methods and availability of datasets for the analysis.

## 1.1 Obstacles to Opinion Mining

Language is idiosyncratic, with nearly infinite possible grammatical variations, misspellings and colloquialisms. All of these aspects make automated analysis of natural language a difficult task. Simple approaches fail when complicated language structures emerge and the use of contextual information is necessary.

The most straight-forward method to classify the piece of text as positive or negative would be to look for "positive" and "negative" words and to decide based on the prevalence of any kind of words. How do we get those sets of words? Pang et al. [2002] showed that it is difficult for people to come up with the right set of words. This does not mean that it is a completely wrong approach. Pang et al. [2002] have also shown that machine learning methods based on this technique can achieve accuracy a little over 80%.

However, systems based on this simple principle would fail for fairly common examples. For instance, at least one of these example sentences:

I like the laptop.
It looks like a laptop.

would be evaluated inaccurately since "like" express positive feelings in the first but in the second it is neutral.

Moreover, people often express sentiments in a subtle way. When reviewing a movie,

"The movie seems to come from a recycling bin."

would be a negative comment without any ostentatiously negative words. A different example would be a sentence like

"The movie has a great cast, the plot seems fascinating,
even the music is good. However, it can't hold up.",

is filled with "positive" words, but holding a negative sentiment overall. The use of irony is sometimes baffling for people too,

"What an original idea, found footage film.
I've never seen a movie like that before."

seems positive, but for people familiar with horror movies it is an obvious use of irony.

## 1.2 Problem Formulation

There are various tasks in sentiment analysis. Pang and Lee [2008] listed an example of problems motivated by different real-world application, using different types of corpora.

A large proportion of such problems is to evaluate a given opinionated piece of text, assumed to be about one single item, as a positive or negative. That would be a *binary classification task*. Positive and negative are not the only categories used. Thomas et al. [2006] classified whether a speech is in support or in opposition to the issue in a debate.

However, it could be a *regression task*, deciding "how positive it is" or a *ranking task ordinal regression*, ordering a set of text from the most positive to the least positive one.

Other instances of problems could be deciding, whether a piece of text is objective or subjective. Mihalcea et al. [2007] suggest that this kind of task is often more difficult than polarity classification, hence the improvement in subjectivity classification will result in improvement of polarity classification as well. Sometimes the objective topic and subjective opinion may interact to an extent that it is desirable to analyze both simultaneously. This is called *joint topic-sentiment analysis*.

## 1.3 Aspect-Based Sentiment Analysis

So far, we have examined only simple examples, addressing only one entity, a movie. However, even movies have several aspects, such as the cast, the music, the plot and many more. In one review, the polarity of the opinion expressed towards any entity or aspect can differ.

"The performance of actors was compelling, the plot not so much."

The above sentence has both positive and negative sentiments but for different aspects. This leaves us with two tasks:

**opinion target identification,** that is to find the aspect in the sentence itself or to classify as some predefined category and

**polarity classification,** deciding whether the aspect is rated positively or negatively.

Generally, methods used for solving those tasks can be

**rule-based** methods, making decisions based on a predefined set of rules, for example motivated by some syntactic rules

**machine learning** methods, when the model that performs the classification is not based on predefined set of rules, rather the rules on which the decision is based is implied from the data i.e. learned

or combined, as in the work by Tamchyna, Fiala, and Veselovská [2008].

## 1.4 Machine Learning Methods

Of course we can carefully analyze data and come up with a set of decision making rules. There are datasets that are so large and complex that they are called "Big data". It would be merely impossible for a human to analyze it all. Instead of torturing people, we would rather present the data and let machines analyze it themselves. This approach is called machine learning.

A system is said to learn from experience with respect to some task, if its performance at the task improves with the experience. We of course need to somehow measure, or grade, the performance to know whether the system is actually learning something. We do it by using a predefined *cost function*. The purpose of the learning, or "training", is to minimize the cost.

Tasks of machine learning are:

**Classification,** where the task is to determine a category to which the data instance belongs to. Models used for classification are usually referred to as *classifiers*,

**Regression,** where the task is to predict a numeric value, based on the presented data. Models used for regression are usually called *predictors* and

**Clustering,** where the task is to automatically find bundles of data which have some common properties.

The data used in machine learning consists of data instances. Data instances have to be presented in a way comprehensible to computer programs. For this purpose, data instances are given as vectors of either numerical or categorical values. Each component of the vector represents some property or some feature of the data instance, hence they are referred to as feature vectors. Sometimes,

along with the feature vector a desired,"the correct", value is given. A feature vector with a given desired output value is called an *example*. Choosing the right features and taking them out of the real data is an exhaustive part of the machine learning process, referred to as *feature extraction*.

Separate sets of data should be used for the training process and the final evaluation of accuracy of the model. It is very important that data instances are split into *training* and *testing* data sets randomly. Otherwise the probabilistic distribution of the data could be different in the two sets, making the evaluation useless for real world examples. The learning methods can be *supervised*, if the desired values are given, or *unsupervised* if they are not given.

There are various models in machine learning. Description of linear and logistic regression, support vector machines, tree based methods, such as decision trees and random forests, clustering methods and many others listed by James et al. [2014]. We further describe models only used in experiments by Tamchyna, Fiala, and Veselovská [2008] and Tamchyna and Veselovská [2016].

### 1.4.1 Cross-Validation

If we take an independent sample for evaluating the model performance, it may happen that the model is not performing as well as on the training set. It fits too well for the training data, hence this is called *overfitting*. It means that the model is not capable of generalization, it "think" too rigidly. The converse problem of *underfitting* is when the model does not fit the training data and the model is generalizing too much.

The goal of machine learning is to find the equilibrium between the problems. Evaluation of the rate of the *underfitting* and *overfitting* is done by cross-validation.

The dataset is randomly split to $k$ equally large subsets. During $k$-fold cross validation, in each of $k$ folds one subset is used as testing data and the rest as training data. The overall accuracy is then estimated as a mean of accuracy achieved in each fold.

### 1.4.2 Conditional Random Fields

In statistics and probability theory, a Markov process is a process satisfying Markov property. To put it simply, it means that predictions of the future of the model can be done solely based on the present state of the process, i.e. the complete process history will not result in a better prediction [Rabiner and Juang, 1986].

Imagine we have $r$ rooms with $n$ urns containing balls of $c$ colors. Every time a room is chosen randomly, a ball is drawn from a randomly chosen urn and then it is put back in the urn. The observed value of the process is just the color of the ball. Draws are completely independent from each other. The process itself is not observed, only the chosen color, thus it is called a *hidden Markov process*.

The state of the process cannot be observed, it is the hidden state. The next hidden state depends only on the current hidden state, not the previous states. The probability distribution of the process can be therefore modeled by a

"chain" of hidden states, where each hidden state has some conditional probability distribution dependent only on the previous hidden state.

Given a sentence, an example of a hidden Markov process could be outputting a set of labels, one for each word, assuming that the probability of the current label is dependent solely on the previous label and the input words. The input words, or simply the sentence, is constant throughout the process.

In the experiment performed by Tamchyna et al. [2008], a model called linear-chain conditional random fields (CRF) was used. It is a statistical model related to a hidden Markov model. However, it is a *discriminative* model; it directly models the conditional probability of a labeling as

$$P(y|x) = \frac{1}{z(x)} \exp\left(\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k f_k(y_t, y_t - 1, t, x)\right),$$

where $z$ is the normalization function

$$z(x) = \sum_{y'} \exp\left(\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k f_k(y'_t, y'_t - 1, t, x)\right),$$

summing over all possible label sequences. The index $t \in \{1, ..., T\}$ corresponds to a position in the sentence, $y_t$ being the label at the position $t$. $K$ is the number of feature or the conventionality of the input vector. Feature function $f_k$ depends on the current label $y_t$, the previous label $y_{t-1}$, the input words $x$ and the associated weights $\lambda_k$. The weights are optimized during the training to maximize the likelihood of the desired labeling.

The prediction is the most probable label, i.e.

$$\hat{y} = \operatorname*{argmin}_y P(y|x)$$

There are methods based on dynamic programming, like Viterbi algorithm [Viterbi, 2006], to efficiently find $\hat{y}$.

### 1.4.3 Logistic Regression

A routine method used for binary classification, also used in the experiment conducted by Tamchyna and Veselovská [2016], is *logistic regression*. It is well described in Introduction to Statistical Learning with Applications in R by James et al. [2014, Chapter 4].

We want to assign a category to each example in the dataset. For binary classification, we will denote the categories by 0 and 1. The predicting function, also called *hypothesis*, has a form of a sigmoidal function, discussed later in Subsection 2.1.1

$$\varphi(x) = \frac{1}{1 + e^{-\Theta^T x}},$$

where $\Theta$ are the parameters of the hypothesis that are learned during the training.

The function outputs the probability of the data instance being of the category denoted by 1. We can choose an arbitrary threshold, although 0.5 is the most commonly used, and classify it as 0, if the probability is lower than the threshold or one if it is greater than or equal to threshold.

The likelihood of the data is the probability of data instances being classified as $y_1, ..., y_n$, where $n$ is the number of data instances and $y_i \in 0, 1$ is the predicted category for the $i$-th data instance $x_i$, is given by the formula

$$\mathcal{L}(y_1, ..., y_n; \Theta, x_1, ...x_n) = \prod_{i=1}^{n} P\left(y_i | x_i, \Theta\right)$$

During the training, the likelihood is maximized. However, it is easier to maximize a summation instead of a possibly very large product, so *log likelihood* function is used as a cost function

$$\begin{aligned} \ell(y_1, ..., y_n; \Theta, x_1, ...x_n) &= \log \mathcal{L}(y_1, ..., y_n; \Theta, x_1, ...x_n) \\ &= \sum_{i=1}^{n} \log P\left(y_i | x_i, \Theta\right). \end{aligned}$$

# 2. Neural Networks

Neural networks are models inspired by the central nervous system of higher animals, humans in particular. Simple nodes, known as "neurons" or "neurodes" are connected together in an artificial network which mimics a real biological neural network.

For practical reasons, the biological approach has been neglected in software implementations, favoring approach based on statistics and signal processing.

## 2.1 Model of a Neuron

A neuron, like in biological neural network, is a fundamental information processing unit of the network.

A model of neuron is characterized by a set of synapses (connections). Each synapse is characterized by its weight. In addition, the neuron also has a given bias (or threshold) and activation (or squashing) function $\varphi$.

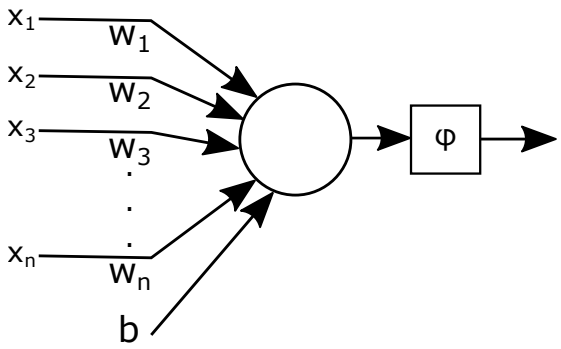


Figure 2.1: A model of neuron

In the Figure 2.1 we see weight of synapses denoted by $w_1, w_2, w_3, ..., w_n$. The input is a vector of numbers $x = (x_1, x_2, x_3, ...x_n)$ and the bias is denoted by $b$. The output of the neuron is computed as

$$y = \varphi\left(b + \sum_{i=1}^{n} x_i w_i\right),$$

where $\varphi$ is some *squashing function* and its argument is called *induced local field*.

### 2.1.1 Squashing Function

The final output depends on the squashing function. One of the most commonly used functions is *threshold function*, which is defined as

$$\varphi(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

However, the prevalent squashing function is the *sigmoid function*. In general, sigmoid function is a bounded differentiable function defined for all real numbers

and has a positive derivative at each point. An example would be a specific case of the logistic function defined by

$$\varphi(x) = \frac{1}{1 + e^{-x}}.$$

This function is used in machine learning enormously, often being referred to simply as *sigmoid function*. Note that this is similar to the loggistic regression model described in Subsection 1.4.3.

Another examples are *signum function*

$$\varphi(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

and the *hyperbolic tangent function*

$$\varphi(x) = \tanh(x).$$

## 2.2 Network Architectures

A single neuron itself is a useful structure, that can be used as a binary classifier for linearly separable data. In order to solve more difficult tasks more neurons are connected together in a network. Neuron output is a single number. If there are more connections starting from one neuron, they all transmit the same number. There are various structures of neural networks, but in general, we may identify three types of architecture of neural networks [Haykin, 2009]

**Single-layer networks** contain only the input and output layer. "Single-layer" refers to the output layer, the input layer is omitted because no computation is performed there. All neurons of the input layer are connected with all neurons of the output layer, but not vice versa. This means that this type of network is strictly feedforward because the connections are oriented from input layer to output layer.

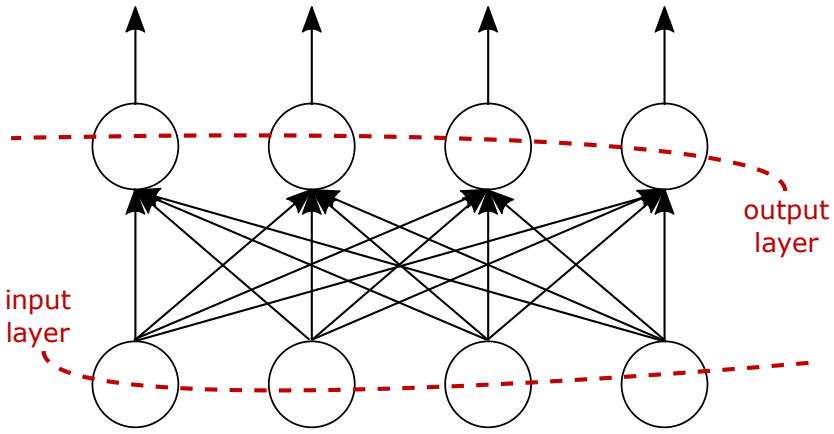


Figure 2.2: An example of a single-layered network

**Multi-layer feed-forward networks** are different from single-layer network by the presence of one or more layers of neurons between the input and output layer. Those layers are called *hidden*, because they cannot be seen directly from neither input nor output layer. The output of the first layer is used as an input for the second layer and so on. Neural networks with only one hidden layer are often called *shallow*, while neural networks with more layers are called *deep neural networks*.

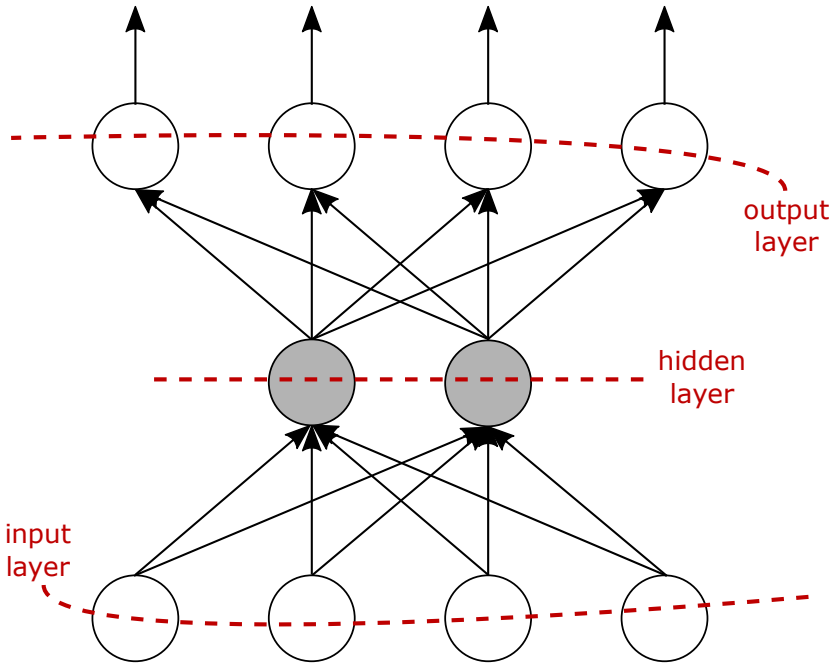


Figure 2.3: An example of a multi-layered feedforward network

Models used by *word2vec*, further discussed in Chapter 3, are a little variation of shallow neural networks.

**Recurrent networks** (or RNN) are distinguished by the fact that they have at least one feedback loop. Feedback loop means that the output of a layer is used again as an input.

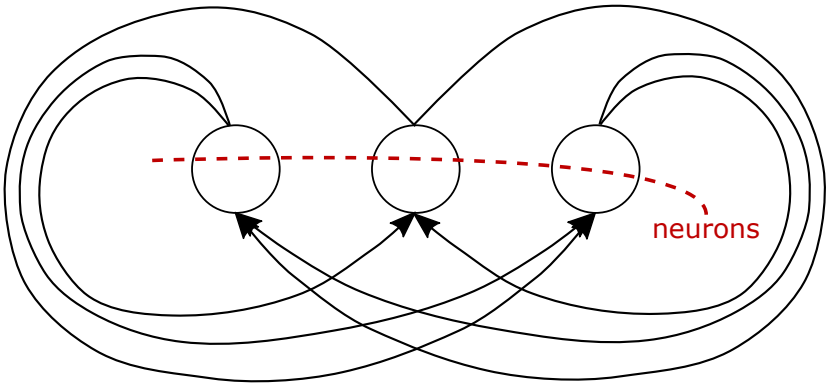


Figure 2.4: An example of a recurrent network

Recurrent networks can be either single-layered (i.e. Hopfield model) or can be multi-layered. Another example of recurrent neural networks are

*long short-term memory* (LSTM) networks proposed by Hochreiter and Schmidhuber [1997] further described in Subsection 2.2.1.

### 2.2.1 Long Short-Term Memory

Now we take a closer look at a specific model of neural networks used in the experiment, specifically the previously mentioned recurrent model called LSTM.

Human thoughts have persistence. People do not just discard what is on their mind and suddenly start to think from scratch. However, this is exactly what feed-forward networks do. Their "mind" is always as if they just woke up. Recurrent neural networks address this issue by preserving the information by feeding it back to the input.

This recurrence may be a bit puzzling at first, but it is actually quite simple. Instead of passing the output back to the input, we can imagine as if the network passed its output to a copy of itself. This way we can imagine the recurrent network as a chain of smaller networks. In fact, chain-like structures, like sentences, movies and real-time handwriting, are exactly the type of structures that recurrent networks are good at processing. Using recurrent networks for speech recognition, language modeling and translating has been a huge success.

Imagine we are trying to predict the next word in a sentence. For instance, the sentence

"The Earth orbits around the ..."

will clearly end in the word "Sun" (we will leave out the possibility of "center of gravity of the solar system" for the sake of simplicity). We did not need to remember the context for too long to complete the sentence. However, when we take a longer sentence such as

"I was born in the USA, although for various reasons,
I don't consider myself an ..."

the necessary context is rather small, but has to be remembered for much longer time. Unfortunately, the gap between the information and the place where it is used as a context can become arbitrarily large. This issue of long-distance dependencies was addressed in the article *Long Short-Term Memory* by Hochreiter and Schmidhuber [1997].

Long short-term memories are suitable for many types of tasks. A great success, in particular, was usage of long short-term memory for automatic recognition of handwriting. Nevertheless, almost all exciting results based on recurrent networks are achieved by using LSTMs.

In simple terms, LSTM remembers a value just by simply re-feeding it over and over again to the memory cell. This principle is called "constant carrousel". This would be useless, if there was not any way to control, whether we want to keep the information, forget it or output it. Of course, there is a way to do so. In LSTM this is achieved by so-called gates.

In the Figure 2.5, $\boldsymbol{x}$ represents the input vector, $c$ is the memory cell and $y$ is the output of the LSTM. The $\otimes$ represents multiplication and circles with a curve inside represent the usage of some squashing (or scaling) function (typically

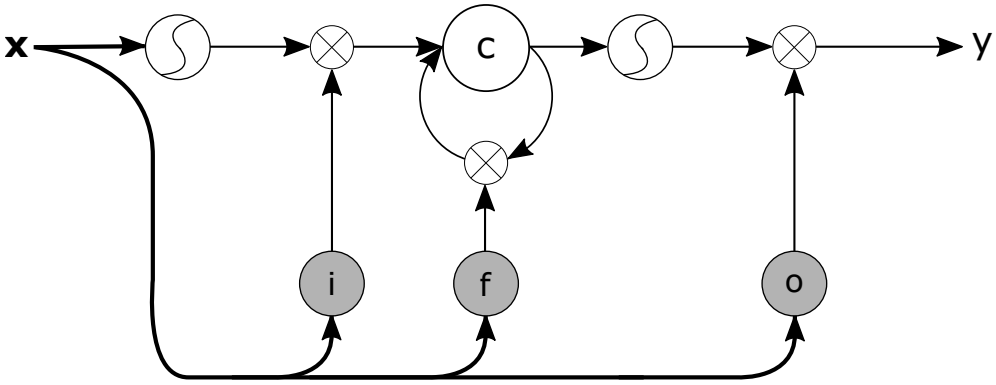


Figure 2.5: A typical implementation of LSTM

the sigmoid or hyperbolic tangent function). The gray nodes are representing the gates, $i$ is the input gate, $f$ is the forget gate and $o$ is the output gate.

The gates are just nodes that output a signal between 0 and 1. This is achieved by the sigmoid function. There are 3 gates:

**input gate** which controls, how much of information is let into the system, with 0 meaning no information is let inside and 1 meaning all the information is let inside the memory cell

**forget gate** which determines, whether we want to forget the information (0) or keep it (1)

**output gate** which controls whether the information should be outputted to the rest of the network (1) or not (0)

At first, the input is fed to all the gates and squashed by sigmoid function (or some other differentiable function). The decision of the *input gate* and the squashed input are then multiplied, which means that the input remains unchanged if the input gate outputs 1 and the input is zeroed if the input gate outputs 0. The output of any gate is just a simple squashed dot product as mentioned in Section 2.1:

$$out_{gate} = \varphi\left(\sum_i w_{gate,i} x_{t,i}\right).$$

The *forget gate* also outputs its decision. The currently remembered vector and the decision of the forget gate are multiplied, meaning that if the output of the forget gate is 1, the pattern remains in memory (is again fed to the input of the memory cell $c_t$) or forgotten, when forget gate outputs 0. Note that the looped input is not put through any squashing function, in order to allow the value to be remembered for a long time without decaying.

Then the remembered value is again squashed by some differentiable function. Finally the output gate decides based on the input $\boldsymbol{x}$ whether the value should be let out of the memory cell to the rest of the system or not. This is done by multiplication, the same way as for the other gates.

In the Figure 2.5, the decision of the gates is based on the input fed also to the memory cell. However, in more general case, the input of the gates and the memory cell does not have to be the same. The rest of the computation remains unchanged even if different inputs for the gates are used.

## 2.3 Training

Neural networks are used in machine learning, which implies that they should be able to learn. More specifically, the network should be able to improve its performance. Similarly to other machine learning models, neural networks are trained on the training data.

There are various methods used to train neural networks. Principally, those methods can be described as

- unsupervised learning methods
- supervised learning methods

The difference is that during supervised learning the desired outputs are used to improve the performance. Supervised learning is also referred to as learning *with a teacher* and unsupervised learning is likewise named learning *without a teacher*. An illustrious example of unsupervised learning method is Hebbian learning. For the purpose of our experiment we will focus on the supervised methods exclusively.

The predominant supervised method for feed-forward neural networks is *back-propagation algorithm*. For recurrent neural networks a modification of the algorithm, called *back-propagation through time*, has been invented.

### 2.3.1 Back-Propagation Algorithm

The name of the algorithm is derived from "back propagation of errors". Many variation of the basic algorithm employing heuristics have been proposed throughout the years. The basic version can be described by following steps:

**Initialize** random weight to all synapses.

**For all** data points in the training set perform

1. Present a data point $x$ and the desired output $d$
2. Compute the actual output $y$ of the network
3. Update the synaptic weight on the layers.
   First update the weights on the output layer,
   then the layer preceding the updated layer and so on.

The weights change in time. The weight of the connection between the neurons $i$ and $j$ at some step is denoted as $w_{i,j}(t)$. The updated weight in the next time step will be

$$w_{i,j}(t+1) = w_{i,j}(t) + \Delta w_{i,j},$$

where $\Delta w_{i,j}$ is the correction applied to the weight.

How do we know what correction we should apply? We need a mathematical way of evaluating the performance of the network. The answer is to present a *cost function*. The cost function, based on the actual and desired output, yields a single number. This is a "grade" the network got for its performance on the task. The purpose of the learning is getting the best grade, which, in mathematical terms, means to *minimize* the cost function.

The correction mentioned above is computed with respect to the learning rate $\eta$, the *local gradient* $\delta_j$ and the input of the neuron $j$, which is the output of the neuron $i$, denoted by $y_i$.

$$\Delta w_{i,j} = \eta \delta_j y_i$$

The use of the local gradient was motivated by gradient descent method, also known as the method of *steepest descent*. A detailed description of the method can be found in the textbook Iterative Methods for Optimization by Kelley [1999].

It is an iterative method for finding the local minimum of a function. The algorithm starts at some point $p_0$. As many times as needed, it moves from $p_i$ to $p_{i+1}$ in the direction of $-\nabla f(p_i)$, where $f$ is the function being minimized.

The local gradient is computed as a first derivation of the *cost function*

$$\mathcal{E} = \frac{1}{2} \sum_{j \in \mathcal{C}} error_j^2,$$

where $error_j$ is simply the difference of the desired and actual output and $\mathcal{C}$ is the set of output neurons. The function is sometimes referred as *energy function* in the context of back-propagation algorithms, but the nuance between cost and energy functions is irrelevant for the purpose of this thesis. The full derivation can be found in the book by Haykin [2009, Chapter 4].

For a neuron on the output layer, the local gradient is

$$\delta_i = error_j \varphi'(v_j),$$

where $v_j$ is the induced local field of the neuron $j$ (explained in Section 2.1). For a hidden neuron we obtain

$$\delta_j = \sum_k \delta_k w_{j,k} \varphi'(v_j)$$

where the index $k$ represents the neurons of layer following the layer with neuron $j$.

Supposing we are using a sigmoidal function

$$\varphi(x) = \frac{1}{1 + e^{-\lambda x}},$$

where $\lambda$ is an adjustable parameter (sometimes referred to as "slope"), we obtain

$$\varphi'(x) = \lambda y_i (a - y_i).$$

Large learning parameter $\eta$ will result in an unstable network. The smaller we make the learning parameter, the smaller are the corrections made to the weights, which means that the training will be "smoother" but much slower. A simple method used to increase the learning weight and avoiding the instability is to use

$$\Delta' w_{i,j} = \eta \delta_j y_i + \alpha \left( w_{i,j}(t) - w_{i,j}(t-1) \right)$$

instead of $\Delta w_{i,j}$ described above.

### 2.3.2 Back-Propagation Through Time

An extension of the back-propagation algorithm described in Subsection 2.3.1 for training recurrent neural networks is called *back-propagation-through-time* algorithm, commonly abbreviated as BPTT. A detailed description of the algorithm is available in the book by Haykin [2009, Chapter 15.7].

The algorithm may be derived from the basic one by *unfolding* the operation of the network into a layered feed-forward network. That means, as we previously described in Subsection 2.2.1, representing the computation as a chain of identical networks, rather than a single network. A computation of a recurrent network (with a single layer) of $n$ steps can be as well represented as a network with $n+1$ layers.

If the whole computations took $n$ epochs (time steps) we can compute the total cost

$$\mathcal{E}_{total} = \frac{1}{2} \sum_{t=1}^{n} \sum_{j \in \mathcal{C}_t} error_{j,n}^2,$$

where $\mathcal{C}_t$ is a set of output neurons in each epoch. Now we can perform the algorithm as proposed by Williams and Peng [1990]. At first a single forward pass through the whole network through all time steps is performed. The complete record of the state of the network is saved.

Then, similarly to the basic algorithm, weights are updated. A slightly different corrections

$$\Delta w_{i,j} = \eta \sum_{t=0}^{n} \delta_{j,n} x_{i,t-1}$$

are applied. The local gradient obtained from the cost function is also similar to the basic version

$$\delta_{j,n} = \begin{cases} \varphi'(v_{j,t})\ error_{j,t} & \text{for } t = n \\ \varphi'(v_{j,n})' \left( error_{j,n} + \sum_{k \in \mathcal{C}_t} \delta_{k,t+1} w_{j,k} \right) & \text{for } 1 \geq t < n \end{cases}$$

Variation of this algorithm is used to train LSTM.

# 3. Word Embeddings and Continuous Bag-of-Words Model

## 3.1 Word Embedding

We may ask ourselves, what precisely is a word. This question may seem trivial, but in fact, it has been a topic of profound discussions. Yet, we can agree that the number of words in any language vocabulary is vast.

As the size of vocabulary is immense, it would be difficult to use the whole dictionary, or rather the number of words in dictionary, for computations. To simplify the computations, the dictionary is mapped to $\mathbb{R}^n$, where $n$ is much smaller than the size of the dictionary. Put simply, every word is represented as a real vector.

It was recently shown by [Tomas Mikolov, 2013c] and [Levy and Goldberg, 2014] that the word vectors capture many linguistic regularities. Performing vector operations on the vectors is quite close to what we would expect.

For example, the vectors can capture morphological dependencies. It was demonstrated that the vector between a singular and a plural form of words are quite similar for all nouns, which means that

$$v_{apple} - v_{apples} \approx v_{family} - v_{families},$$

where $v_{apple}$ is a vector representation of the word *apple* along with others.

Nevertheless, word embeddings are able to capture deeper dependencies, such as the similarity of the meaning between *king* and *queen*. More precisely, we can approximate the vector representation of the word "queen" just by using vector representations of the words "king", "man" and "woman" as

$$v_{king} - v_{man} + v_{woman} \approx v_{queen}.$$

The motivation for identifying opinion target is that words used to describe some entity should be somehow similar.

*Word2vec* is a group of models that are used to generate the word embeddings. Various methods for generating the embeddings exist, such as dimensionality reduction on the word co-occurrence matrix. We will focus on the methods based on neural networks. There are two main models used for generating the vectors. *Skip-gram* model proposed by Mikolov et al. [2013b] and *continuous bag-of-words* (CBOW) model proposed by Mikolov et al. [2013a]. The skip-gram model is more suitable for infrequent words, on the other hand, it is much slower. Since we can expect that customer reviews are not filled with precarious words, for the purpose of the experiment we chose CBOW.

## 3.2 Continuous Bag-of-Words Model

Now we shall enlighten how the vector representations of word are constructed. Several methods to achieve this goal exist. In this section, we will focus on how it is obtained by the use of neural networks.

Imagine we have a large dictionary of size $n$, where $n$ is a large number, for example The Dictionary of Contemporary Czech contains approximately 200000 words. For computations we would prefer to work with real vectors of dimension $m$. How do we obtain those representations by the use of neural networks?

To capture the previously described dependencies, we have to somehow use the context in which the words are used. A bigram is a sequence of two elements, in our case two following words. A sequence of three words would be trigram, generally a sequence of $n$ words is called $n$-gram. Given the first word, we would like to predict the following word. To accomplish the task we will use a simple multi-layer neural network shown in Figure 3.1, a much larger version of the network from Figure 2.3.

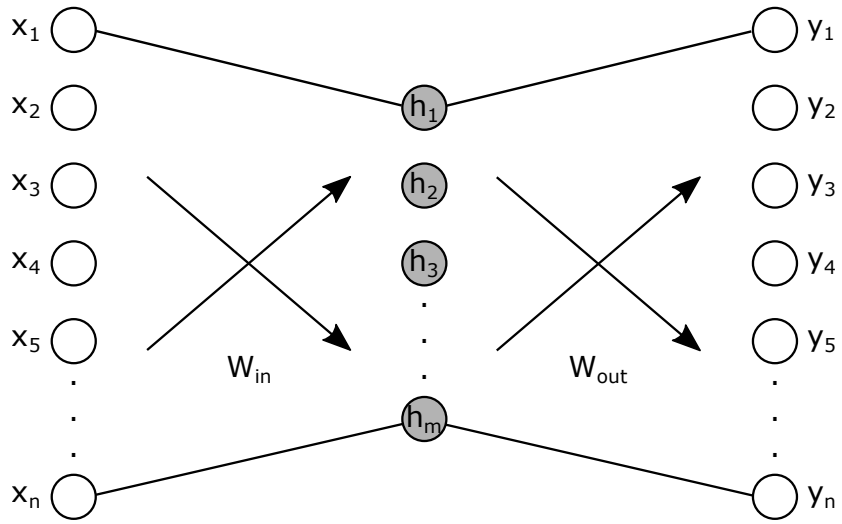


Figure 3.1: A basic example of the CBOW model for the context of one word

The input and the output layer will be of size $n$. The hidden layer will be of size $m$. Each neuron would represent one word, so the input would be a vector of dimension $n$ consisting of zeros and a single 1, which would represent the first word. We would like to obtain a similar output, a vector containing only zeros and a single 1, which would represent a prediction of the word. In that case, we would predict a single word. In authentic examples, there are usually more than one possibilities. Rather than a single one, we would prefer a vector that would contain real numbers between and 0 and 1, representing the *probability* of the occurrence of the word.

By training a network of that type we would obtain some weights between neurons of the input layer and the output layer. If we denote the weight of synapse between $i$-th neuron of the input layer and the $j$-th neuron of the hidden layer by $w_{i,j}$, we obtain a matrix $W \in \mathbb{R}^{m \times n}$. If the input is $k$-th word, the signal

that arrives to the hidden layer can be expressed as

$$u_k^T = \begin{pmatrix} 0 & 0 & \cdots & 1 & \cdots & 0 \end{pmatrix} \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1} & w_{k,2} & \cdots & w_{k,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{pmatrix}$$
$$= \begin{pmatrix} w_{k,1} & w_{k,2} & \cdots & w_{k,n} \end{pmatrix},$$

which means, that the vector $u_k$ is a representation of the $k$-th word.

Unfortunately, we have attained two matrices, one is the matrix $W_{in}$ representing the weights between the input and hidden layer, the other matrix $W_{out}$ representing the weights between hidden and output layer. Now we have two sets of vector representations, input vectors and output vectors. The two sets of vector representations are usually combined by summation; nonetheless, concatenation is also used.

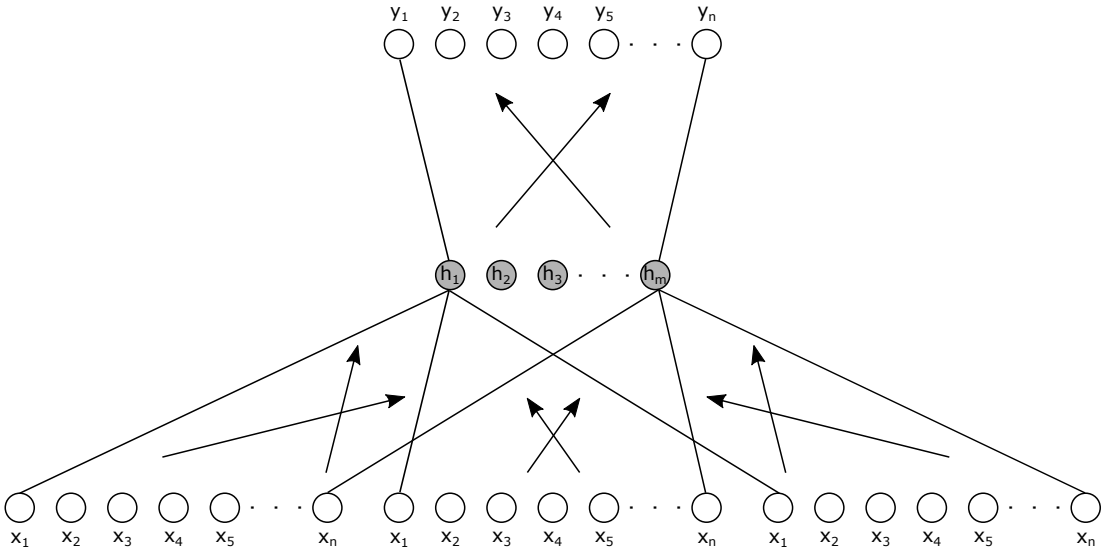


Figure 3.2: CBOW model using context of size 3

Besides, for real sentences, a considerably broader context is required. We used only one word to predict the other, in other words we have used context of one word. Usually we will need a broader context. Mikolov et al. [2013a] achieved the best results for using a context of 8 words, 4 word preceding and 4 words following the predicted word. In the CBOW model, this is accomplished by using more copies of the input layer, where each copy represents one given word. An example of CBOW model with a context of 3 words is shown in Figure 3.2.

When computing the output of the hidden layer, an average of the all inputs is used. The reason, why it is called "bag-of-words" model, is that the result are independent of the word order, since simple average of the inputs is used.

### 3.2.1 Training

As we mentioned earlier, the goal of the CBOW model is to output the probabilities for each word, formally, the probability $P(word \mid context)$. Mikolov et al. [2013a] achieved this by using softmax function. That is

$$P(word_k \mid context) \approx \frac{e^{output_k}}{\sum_{i=1}^{n} e^{output_i}},$$

where $output_k$ is the output of the $k$-th neuron, which is the prediction of the probability of $k$-th word being suitable for the *context*.

During the training, for a given bigram, trigram or generally $c$-gram, where $c$ is the number of words in the context, we can obtain the desired output - a vector with all zeros and a single 1 on the position $k$, indicating that $k$-th word of the dictionary should have been predicted. Then we can simply train the network by using back-propagation algorithm, described in Subsection 2.3.1.

# 4. Dataset

In the experiment we used data obtained from Wikipedia and the dataset Aspect-Term Annotated Customer Reviews in Czech[1], that contains reviews of electronic devices obtained from an established Czech online shop.

## 4.1 Wikipedia Dump

As the training data for *word2vec*[2] we used the current dump of Wikipedia. Wikimedia Foundation publishes data dumps from Wikipedia (and other Wikimedia projects) on a regular basis, for English and Czech Wikipedia, once a month.

The dump is a single large `.xml` file containing text and metadata of all current Wikipedia pages. An article from Czech Wikipedia is contained in a `page` element.

```
  <page>
    <title>Strojové učení</title>
    <ns>0</ns>
    <id>94861</id>
    <revision>
      <id>13754175</id>
      <parentid>13754173</parentid>
      <timestamp>2016-05-29T19:40:13Z</timestamp>
      <contributor>
        <username>Ch!p</username>
        <id>8025</id>
      </contributor>
      <minor />
      <model>wikitext</model>
      <format>text/x-wiki</format>
      <text xml:space="preserve">'''Strojové učení''' je podoblastí
[[umělá inteligence|umělé inteligence]], zabývající se...
```

We use WikiExtractor[3] to obtain only the text from the dump, with no additional metadata.

```
Strojové učení je podoblastí umělé inteligence, zabývající se...
```

The recommendation is to have one sentence per line, unrelated data separated with an empty line and also discarding the punctuation. Replacing all numbers with special `NUM` token may also improve the embeddings. We will compare the results of embeddings obtained from a text with numbers and text with `NUM` tokens. We preprocess the data by using our script `preprocessing.py` (see Appendix A).

Converting all letters to lower case is not necessary and depends on the application. It may be useful to distinguish *Apple* from *apple*. A model trained on

[1] The dataset is available at
`https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-1507`

[2] Availible at `https://pypi.python.org/pypi/gensim/0.13.1`

[3] WikiExtractor is available at `https://github.com/attardi/wikiextractor`

data without converting to lower case characters recognize words “procesor” and “Procesor” as two different words.

```
>>> m.most_similar('procesor', topn=5)
[('mikroprocesor', 0.8615756034851074),
 ('čip', 0.8593242168426514),
 ('počítač', 0.8330967426300049),
 ('CPU', 0.8149457573890686),
 ('Procesor', 0.8145838975906372)]
```

Above we can see the closest words to the word “procesor”. The distance is measured simply as the *cosine* of the angle between the two vectors. There is no difference in meaning of “procesor” and “Procesor” but they are not the closest ones.

Therefore, for the purpose of the experiment it would be more useful to use only lower cased data. The conversion to lower case is included in our script.

## 4.2 Czech Dataset

The original dataset contains 1000 positive and 1000 negative short segments from reviews with manually tagged targets. It also contains a hundred of positive and a hundred of negative reviews.

Bellow is an example of the segment from the dataset where the opinion target is the product itself, represented by the word “Kabel” (cable).

```
<positive_summary id="1000000358">
<target>Kabel</target>, který funguje - víc netřeba
</positive_summary>
```

### 4.2.1 Dataset Preprocessing

The original dataset contained examples not only in Czech language, but also in Slovak and a few in English. Czech and Slovak languages are closely related. Even for foreigners who learned Czech, Slovak language is intelligible to some extent. Despite that, using a mixed data set for the analysis may confound the model.

*How do we combine the two sets of word embeddings for the two languages in order to achieve the best results?* Despite being an interesting question, it shall remain unanswered by this thesis. In our experiment we will focus exclusively on the Czech data examples.

By removing the reviews in Slovak, we obtain a dataset of

- 852 short negative examples,
- 85 long negative examples,
- 924 short positive examples and
- 74 long positive examples,

which means we have lost about 12.05% of data examples.

"The data come from the 'wild' , they are quite noisy," suggest Tamchyna, Fiala, and Veselovská [2008], "many segments are written without accents, reducing the benefits of morphological analysis..." This would have fatal consequences for our model, therefore, as Tamchyna, Fiala, and Veselovská [2008] proposed, we use spell-checker to fill in the accents.

The point is only to complete the diacritical signs, not to change the data. The examples contain many slang words, alternative spellings, which are not considered correct but are common, e.g. *"potřebuju"* instead of *"potřebuji"* or *"dizajn"* instead of *"design"*. We preserve those vernacular aspects of the segments. However, because of the correction being done by a single person, the author of this thesis, it may happen, that some words have not been given their accents or have been corrected "too much".

Since the Czech and Slovak languages are intertwined, sometimes, especially for short samples, it is not possible to decide whether user typed without punctuation, made a typo or actually wrote in Slovak. Therefore, some segments may have been accidentally translated by the use of the spell-checker.

The long reviews are not suitable for training our model. Only a scarce number of long examples have been provided, which would require many iterations over the data. Furthermore, long reviews are usually detailed, mentioning almost all aspects, therefore the model would probably learn a simple rule such as: "if it is long, then it is in all categories". The experiment with recurrent neural networks by Tamchyna and Veselovská [2016] was also performed on short samples. For the reasons described above, we will focus on the short samples.

Since only a limited amount of short samples was provided, we replenish the short samples, thus obtaining a new version of the dataset (see Appendix B). We download `.html` pages with reviews of randomly selected products from a Czech e-shop[4]. Using our bash script `preprocess_reviews.sh` (see Appendix A) we extract the reviews consisting of at least six consequential non-whitespace characters, since many of the reviews are just simple *"nic"* (nothing) or *"žádné"* (none) already sufficiently contained in the original data.

The purpose of the experiment in this thesis is to identify categories of the opinion target. Therefore, it was necessary to choose the right categories and assign the desired categories to the examples. We decided to use categories:

**service** (S) provided by the e-shop

**general** (G) properties of the device

**functionality** (F) and performance of the device

**build quality** (B) and build properties of the device

**price** (P)

**design** (D) or the aesthetic aspect of the device

[4] `www.alza.cz`

# 5. Experiment

## 5.1 Previous Work

This thesis stands on previous experiments, namely the experiment with the same dataset conducted by Tamchyna, Fiala, and Veselovská [2008] and the experiment with RNN learning model performed by Tamchyna and Veselovská [2016].

Our goal is to compare the results of those two experiments. The comparison is only indicative, since the dataset and the approach have been slightly modified.

### 5.1.1 Experiment with the Czech Dataset

Tamchyna and Veselovská [2016] used a probabilistic model, described previously in Subsection 1.4.2. They also obtained better results by employing features based on morphological analysis, subjectivity lexicon and syntactic rules.

At first, surface features were used, that is the words as they are written — a sequence of characters. They extracted all bigrams and trigrams from a window consisting of the current word, two preceding and two following words.

Morpho-syntactic features were derived from *lemma*, *morphological tag* and *analytic function*, which decides whether the word is a subject, predicate, etc.

Subjectivity lexicon is a list of subjectivity clues for sentiment analysis. Czech SubLex 1.0[1] was used to obtain a feature indicating whether a word was marked as subjective. Finally, rule features were extracted for each rule, indicating whether the rule was applicable to the word.

Unlike the latter experiment, the task was a labeling task. The task was to identify the opinion target in the example text itself. The results of the experiment are shown in Table 5.1

| **Features** | **Precision** | **Recall** | **F-measure** |
|---|---|---|---|
| Surface | 85.22 | 36.85 | 51.45 |
| + Morpho-syntactic | 75.88 | 54.17 | 63.21 |
| + Subjectivity lexicon | 78.19 | 55.09 | 64.64 |
| + Rule | 76.54 | 57.69 | 65.79 |

Table 5.1: Precision, recall and F-measure obtained in the experiment with Czech dataset with short reviews.

Several performance measures of the system were used in the experiment. Let us briefly explain the used methods of performance measure.

**Precision** describes how many of the selected elements (in this case words, or rather tokens) have been selected correctly. These are "true positives".

The elements selected incorrectly are called "false positives". Likewise, the elements that have not been selected are called "true negatives" if they

[1] Available at `https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-1507`

should *not* have be selected. "False negatives" are the elements that should have been selected but were not.

This is a useful performance measure. However, if the classifier selected only one item and it was selected correctly, it would be 100% but it would be useless, because it selected only a small number of elements that should have been selected. That is why *recall* (also known as *sensitivity*) is used too.

**Recall** expresses how many elements have been selected out of all that should have been selected. Again, a model that selects all elements would have 100% recall. Therefore, we need a performance measure that would depict the trade-of between precision and recall.

**F-measure** (or $F_1$ score) is the harmonic mean of precision and recall given by the formula:

$$F_1 = 2\,\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

### 5.1.2 Recurrent Neural Network for Sentence Classification

In this experiment, Tamchyna and Veselovská [2016] used the model of a neural network shown on Figure 5.1

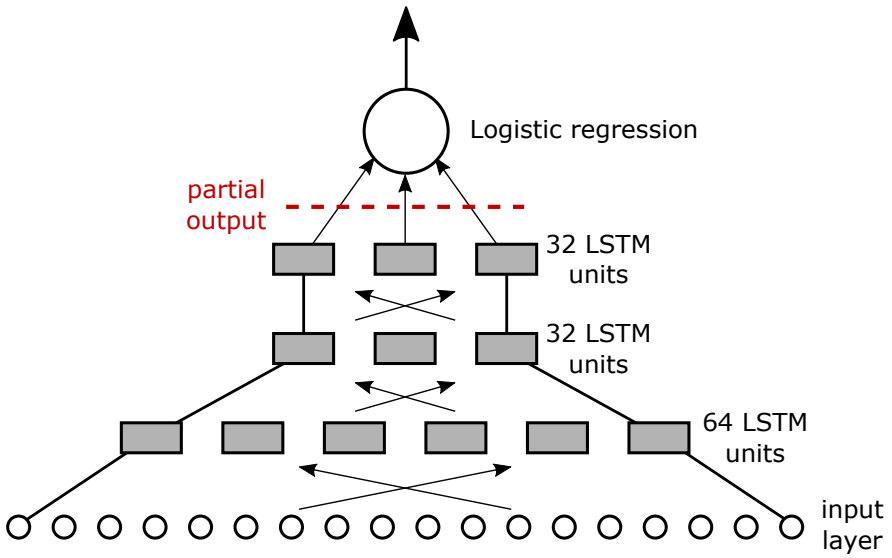


Figure 5.1: The model used in the previous experiment

A single model is fed with the whole sample. More precisely, it gets embeddings obtained from pretrained Word2Vec, one by one. If a word does not have an embedding, it is simply ignored. The dimension of the embeddings is 200. A separate network is used for each category. During reading the sample, the partial output is ignored. This may seem as a huge drawback, but all LSTM units are RNNs, which means that they are trained by BPTT described in Subsection 2.3.2. Therefore, the last output is dependent on the previous outputs

and it is taken into account during the training. After reading the whole sample, logistic regression layer outputs its decision, based on the last partial output.

The task of the experiment was multi-label classification, which means that multiple classes (or "labels") could be assigned to the same data sample.

There are different approaches to measurement of precision and recall, listed by Sorower [2010]. The model could be "partially correct". Of course, we could ignore those partially correct predictions and compute so-called "exact-match ratio".

However, the measurement used in the experiment is to compute precision as

$$\text{precision} = \frac{1}{n}\sum_{i=1}^{n}\frac{|\text{true} \cap \text{prediction}|}{|\text{prediction}|}$$

and recall as

$$\text{recall} = \frac{1}{n}\sum_{i=1}^{n}\frac{|\text{true} \cap \text{prediction}|}{|\text{true}|},$$

where *true* is the set of correct labels and *prediction* is the set of predicted labels for each on $n$ data examples.

The same task was performed for 8 separate datasets. Six of them were restaurant reviews in six different languages — Dutch, English, French, Russian, Russian, Spanish and Turkish. In addition, hotel reviews in Arabic, and consumer electronics (laptops) reviews in English were used.

The acquired results vary greatly, for some datasets the model did not even outperform the baseline, for others the model performed remarkably good (see Table 5.2).

| **Language** | **F-measure** | **Language** | **F-measure** |
|---|---|---|---|
| English (restaurants) | 59.30 | Dutch | 55.03 |
| English (laptops) | 38.26 | Russian | 64.83 |
| Spanish | 58.81 | Turkish | 61.03 |
| French | 50.84 | Arabic (hotels) | 52.59 |

Table 5.2: Best F-measure obtained for all datasets

## 5.2 Experiment with the Original Model

The model was evaluated for various languages, but Czech was not one of them. Our ambition is to examine the model's behavior for Czech data — more specifically consumer electronics review in Czech.

### 5.2.1 Word Embeddings

We use our script `train_embeddings.py` (see Appendix A) to train the word embeddings using the preprocessed dump of Wikipedia as the training data (see Section 4.1). The obtained vocabulary is approximately half a million words.

The dimension of the embeddings is 200, the same as in the original experiment. We used word frequency threshold of 5, which means ignoring the words that occurred less than 5 times in the text. The size of context window we used is also 5, as the authors recommend for CBOW.[2]

When the model is trained, we can experiment with the embeddings. Let us try the example from Chapter 3,

$$v_{king} - v_{man} + v_{woman} \approx v_{queen},$$

using Czech equivalents.

```
>>> w2v.most_similar(positive=['král', 'žena'], negative=['muž'],
    topn=1)
[('královna', 0.6936340928077698)]
```

That was a success. We can witness that the model also learned some common geographical knowledge, such as associate the capital with the country

```
>>> w2v.most_similar(positive=['paříž', 'německo'],
    negative=['francie'], topn=1)
[('berlín', 0.6698291301727295)]
>>>  w2v.most_similar(positive=['paříž', 'slovinsko'],
     negative=['francie'], topn=1)
[('lublaň', 0.5795571208000183)]
```

However, it is not perfect. The correct answer is not always the closest one.

```
>>> w2v.most_similar(positive=['paříž', 'itálie'],
    negative=['francie'], topn=3)
[('vídeň', 0.6771950721740723),
 ('neapol', 0.6644973158836365),
 ('řím', 0.6556979417800903),]
```

In this specific case it could have been caused by common usage of "Řím" as Roman Empire.

We can also use the model to obtain some political knowledge. For example, knowing the president of the Unites States, we can ask for the president of Russia

```
>>> w2v.most_similar(positive=['obama', 'rusko'],
    negative=['usa'], topn=1)
[('putin', 0.5721254348754883)]
```

Or knowing the first name of the contemporary president, we can acquire the first name of a past president

```
>>> w2v.most_similar(positive=['ronald', 'obama'],
    negative=['reagan'], topn=1)
[('barack', 0.766093909740448)]
```

All the examples above work with both embeddings, the embeddings with numbers and embeddings with `NUM` token. The distances vary a little, the above-listed distances were obtained by the use of embeddings with `NUM` token.

[2]Google Code Archive; [accessed 2016 July 13], `https://code.google.com/archive/p/word2vec/`

### 5.2.2 Baseline

For comparison, we can use the previous results (see Table 5.1 and Table 5.2) and a simple baseline models.

The most common combination of labels is the category "G" alone. Overall, the most frequent labels are "G", "B" and "F".

| **"Predicted" labels** | **F-measure** |
|---|---|
| GF | 45.65 |
| BFG | 45.58 |
| BG | 43.99 |
| G | 43.74 |
| BDFGPS | 31.21 |

Table 5.3: F-measures obtained for a trivial model with a fixed set of "predicted" labels

We tried the trivial models that simply chose the same labels for all data instances and measured their F-measure over the whole dataset (see Table 5.3). We chose the best F-measure obtained by a trivial classifier, **45.65%**.

### 5.2.3 Results of the Original Model

We perform the evaluation of the model described by Tamchyna and Veselovská [2016] with no modification (Figure 5.1) by a 5-fold cross-validation (see Subsection 1.4.1).

The whole computation is performed by our implementation of the model (see `model.py` in Appendix A).

We compare two types of embeddings - word embeddings recognizing numbers and embeddings only recognizing a numeric token. The evaluation is performed on only 2000 considerably complex examples. Therefore, trying more iterations through the whole dataset is reasonable.

The word embeddings including numbers are more up-and-coming, so we will focus on those in all later experiments. This is understandable as the dataset consists of reviews of electronic devices, which inevitably contains numbers. The highest achieved F-measure was **47.27%** for **100 iterations** (see Figure 5.2. The model performed better than the trivial model and the performance is comparable to the models in the original experiment.

## 5.3 Modified Model

As we mentioned previously in Subsection 5.1.2, there are some weak points of the model, some of them pointed out by Tamchyna and Veselovská [2016]. In the following sections we will try to address those issues.

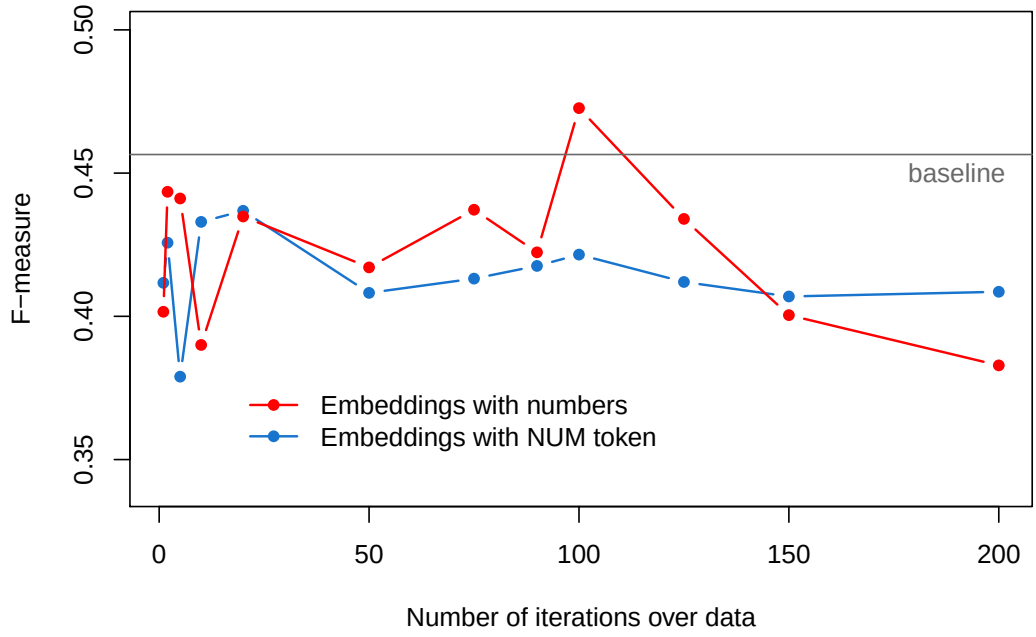


Figure 5.2: F-measures obtained for different number of iterations

### 5.3.1 Decision Threshold

At first, let us have a look on the performance measures of model trained with around 100 iterations (see Figure 5.3).

We can observe that the precision is generally higher that the recall. The model is having more difficulties with picking a label than picking the correct one.

```
Desired output  : F
Probability of B: 0.00269699
Probability of D: 0.00201485
Probability of F: 0.472179
Probability of G: 0.0234346
Probability of P: 0.00218663
Probability of S: 0.00198522
```

```
Desired output  : G
Probability of B: 0.000408143
Probability of D: 0.000288665
Probability of F: 0.000708789
Probability of G: 0.00566146
Probability of P: 0.00130722
Probability of S: 0.000280321
```

The above examples show that even though some examples are confounding to the model, it often chooses the correct one as the most probable.

While the first example may tempt us to just shift the decision threshold a bit, the second suggests that it may not be enough. Tuning the threshold is

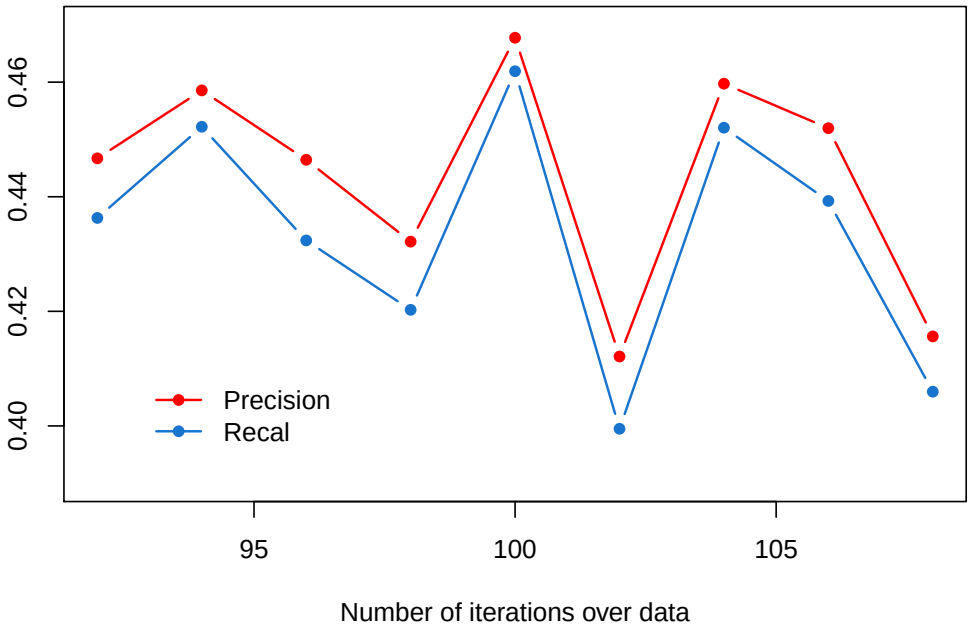


Figure 5.3: Precision and recall for different numbers of iterations

always possible. However, it is time demanding. In our experiment, 1768 out of 2000 examples (or 88.4%) has only a single desired label. Considering that, we propose a simpler solution that will work in both above cases.

Picking no label is doing no good to the model's performance. The way how the performance measures are computed (see Subsection 5.1.2) leaves the model with zero award for not choosing anything as well as choosing the wrong label. This leads us to a simple solution. If no labels have been chosen, choose the most probable one.

By this simple patch, we obtained the best F-measure of **64.78%** for **150 iterations**, which is approximately the same as the best performance in the experiment - performance on Russian data (see Table 5.2).

### 5.3.2 Reducing the Number of Models

As we mentioned previously in Subsection 5.1.2, we train a separate model for each category. Tamchyna and Veselovská [2016] suggested a hypothesis that by using a single model with only a separate logistic regression unit for each category we reduce the time required for the training. It may also improve the performance as that architecture allows the parameters to be somewhat shared.

We implement and test their hypothesis. It turned out that the training indeed took less time. The G category is prevalent in our dataset. It is present in almost half of the samples (45.15%). We found out that the classifiers were rather "fighting" than cooperating and for all number of iterations we tried, (1, 5, 10, 20, 50, 75, 100, 600, 2400) the classifier was able to predict only the prevalent class.

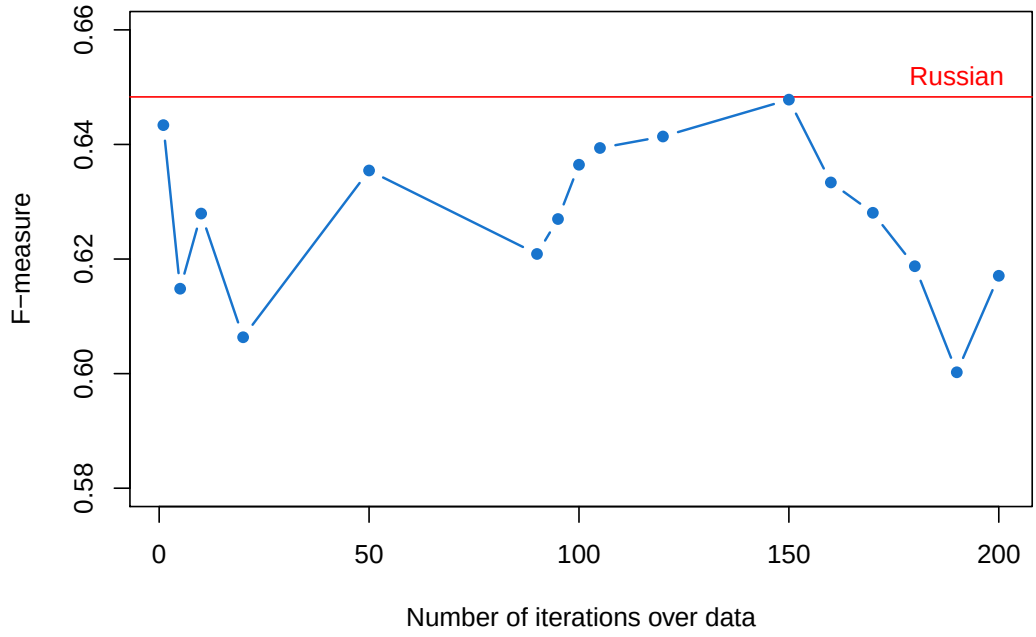


Figure 5.4: F-measures obtained for different number of iterations

It may work for an evenly proportioned dataset. The definition of this model is contained in our implementation of the model in the script `experiment.py`; however, `model.py` must be slightly modified in order to use it (see Appendix A).

### 5.3.3 Max-Pooling

In Subsection 5.1.2 we mentioned that the partial output is ignored while the sentence is being read. Using also the partial output could improve the performance. On the other hand, it may also not, as the partial output is already taken in consideration by using BPTT.

For this purpose pooling methods are used. Average pooling was largely used in the past. Recently, max-pooling became popular due to achieved results. More precisely, we propose to use max-pooling over time. That is simply element-wise maximum of all partial outputs.

Our motivation for using max-pooling is that higher activation means higher importance. By using max-pooling over time, the highest activation is used more during the training. The obtained results, shown in Figure 5.5 show that the max-pooling have produced slightly better performance. The highest achieved F-measure was **66.93%** for only 20 iterations. The evaluation was performed by the scrip `model.py` using `-m` option (see Appendix A)

### 5.3.4 Tuning the Number of Long Short-Term Memories

By using max-pooling we obtained slightly better results for only 20 iterations over the dataset. Smaller number of iterations allows us to test what number of

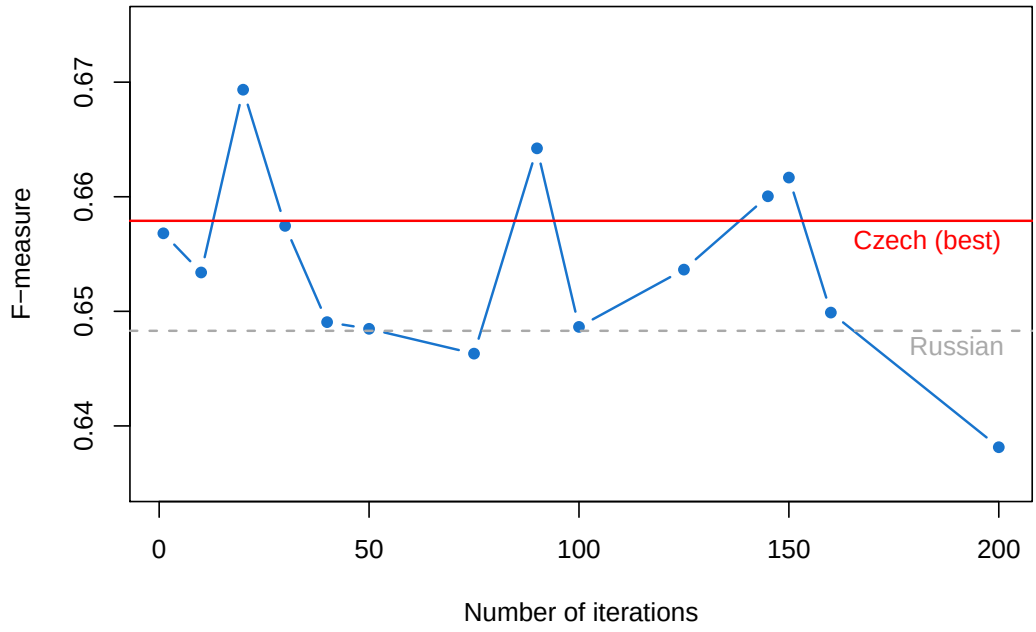


Figure 5.5: The model used in the previous experiment

LSTM cells in each layer results in the best performance.

We tried different number of LSTMs in each layer and discovered that the relation between the number of units and performance is quite unpredictable (see Table 5.4).

| **1st layer** | **2nd layer** | **3rd layer** | **F-measure** |
|---|---|---|---|
| 64 | 32 | 16 | 63.82 |
| 64 | 64 | 32 | 64.59 |
| 128 | 64 | 32 | 64.11 |
| 100 | 50 | 25 | 65.5 |
| 90 | 50 | 25 | 65.49 |
| 80 | 40 | 25 | 66.81 |
| 78 | 41 | 26 | 63.81 |
| 66 | 33 | 27 | 63.83 |

Table 5.4: Several examples of F-measures obtained for models with different number of LSTMs in hidden layers.

We have achieved some results close to our best result (66.93%), but we have not outperformed it.

# Conclusion

The objective of the thesis is to identify the opinion target in short reviews of electronic devices. We used six labels to describe different opinion targets. Each review has one or more labels.

We prepared the dataset and evaluated the model's performance with 5-fold cross validation. Firstly, we removed reviews in Slovak and replenished the dataset with new Czech examples. Secondly, we filled in diacritical signs and assigned a category to each data instance.

We preprocessed the current dump of Czech Wikipedia and trained embeddings. We tested two sets of word embeddings — one trained with using numbers as context information and one with the numbers replaced with a `NUM` token. The results have shown that embeddings with numbers are better for our specific experiment.

We implemented a model of neural network described by Tamchyna and Veselovská [2016] and measured its performance for various number of iterations. The highest achieved F-measure was **47.27%**, which is not very high, since our best trivial classifier scored 45.65%.

Therefore, we tried to achieve better results by modifying the way how the final decision is made. This was a successful attempt and we obtained F-measure of **64.78%**, very close to the best results in the original experiment (64.83%).

We also tried to use a single modified classifier as Tamchyna and Veselovská [2016] suggested for future work. We observed significantly shorter training time but very poor performance. In fact, the model behaved like the trivial baseline classifier — only predicting the most frequent label.

We also introduced max-pooling into the model in order to emphasize higher activation while the model "reads" the example. In resulted in a minor improvement of the performance. More importantly, we obtained the highest F-measure of **66.93%** for only 20 iterations. That is slightly higher than the F-measure of the previously used probabilistic models (65.79%) on the same dataset before we modified it.

We tried to change the numbers of LSTM units in different layers and observed that the relation between the number of units and F-measure is quite unpredictable. We have not managed to achieve better results.

## Future Work

We used the same structure for all classifiers. Trying different numbers of units for different classifiers may result in higher overall performance. Testing the models with a modified number of LSTM units for different numbers of iterations may also increase the performance.

We did not experiment with the embedding dimension and context window size, since the training of the CBOW model is time consuming. In addition, the time increases with the increasing embedding and window size.

Similarly to the problems of overfitting and underfitting, there is some ratio of window and embedding size that yields the best results. It would be interesting

to use higher computational capacities to measure the performance for various size of the embeddings and the context window in order to find the ideal ratio.

As we have shown in Section 5.3.1, the corpora used for training the embeddings does influence the performance of the model. We used the current dump of Wikipedia as the training data for the embeddings. However, since the reviews often mention technical specifications of the devices, adding some technical text to the corpora may boost the performance.

The dataset is a crucial ingredient of any machine learning experiment. We suggest that more data instances of non-trivial length are collected and the categories are verified by more people. Also during our experiment, the model often lost vital information because a colloquial expression was used. This issue could be addressed by lemmatization or using some slang dictionary.

Despite many possible improvements, the usage of neural networks for ABSA seems promising. Although the model performed differently for individual languages, the experiment does not require any knowledge of the language. This principle is vital, especially in the contemporary era where languages evolve faster than ever before.

# Bibliography


Simon Haykin. *Neural Networks and Learning Machines*. Third Edition. Pearson, Hamilton, Ontario, Canada, 2009. ISBN 978-0-13-147139-9.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735—-1780, 1997.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014.

C. T. Kelley. *Iterative Methods for Optimization*. Frontiers in applied mathematics. SIAM, Philadelphia, 1999. ISBN 0-89871-433-8.

Omer Levy and Yoav Goldberg. Linguistic regularities in sparse and explicit word representations. In *CoNLL*, page 171–180, 2014.

Rada Mihalcea, Carmen Banea, and Janyce Wiebe. Learning multilingual subjective language via cross-lingual projections. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, 2007.

T. Mikolov, K. Chen, G. S. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, page 3111–3119, 2013b.

Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1-2):1–135, January 2008. ISSN 1554-0669.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. In *Empirical Methods in Natural Language Processing*, pages 79–86, 2002.

L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSp Magazine*, 1986.

Mohammad S Sorower. A literature survey on algorithms for multi-label learning. Technical report, 2010.

Aleš Tamchyna, Ondřej Fiala, and Kateřina Veselovská. Czech aspect-based sentiment analysis: A new dataset and preliminary results. In *Proceedings of the 15th conference ITAT 2015: Slovenskočeský NLP workshop (SloNLP 2015)*, pages 95–99. Association for Computational Linguistics, 2008. ISBN 978-1515120650.

Aleš Tamchyna and Kateřina Veselovská. UFAL at semeval-2016 task 5: Recurrent neural networks for sentence classification. 2016.

Matt Thomas, Bo Pang, and Lillian Lee. Get out the vote: Determining support or opposition from Congressional floor-debate transcripts. In *Proceedings of EMNLP*, pages 327–335, 2006.

Geoffrey Zweig Tomas Mikolov, Wen-tau Yih. Linguistic regularities in continuous space word representations. Association for Computational Linguistics, May 2013c.

A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inf. Theor.*, 13(2):260–269, September 2006. ISSN 0018-9448. doi: 10.1109/TIT.1967.1054010.

Ronald J. Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 2:490–501, 1990.

# List of Figures


2.1 A model of neuron . . . . . 10
2.2 An example of a single-layered network . . . . . 11
2.3 An example of a multi-layered feedforward network . . . . . 12
2.4 An example of a recurrent network . . . . . 12
2.5 A typical implementation of LSTM . . . . . 14

3.1 A basic example of the CBOW model for the context of one word 19
3.2 CBOW model using context of size 3 . . . . . 20

5.1 The model used in the previous experiment . . . . . 26
5.2 F-measures obtained for different number of iterations . . . . . 30
5.3 Precision and recall for different numbers of iterations . . . . . 31
5.4 F-measures obtained for different number of iterations . . . . . 32
5.5 The model used in the previous experiment . . . . . 33

# List of Tables

5.1 Precision, recall and F-measure obtained in the experiment with Czech dataset with short reviews. . . . . 25
5.2 Best F-measure obtained for all datasets . . . . . 27
5.3 F-measures obtained for a trivial model with a fixed set of "predicted" labels . . . . . 29
5.4 Several examples of F-measures obtained for models with different number of LSTMs in hidden layers. . . . . 33

# List of Abbreviations

**ABSA** Aspec-Based Sentiment Analysis

**BPTT** Back-Propagation-Through-Time algorithm, depicted in Subsection 2.3.2

**CBOW** Continuous Bag of Words model, described in Section 3.2

**CRF** Conditional Random Fields, described in Subsection 1.4.2

**LSTM** Long Short-Term Memory, described in Subsection 2.2.1

**RNN** Recurrent Neural Network

# Attachments

## Appendix A

The attached DVD contains in the **scripts** directory the following scripts:

**experiment.py** contains class and function definitions

**preprocess.py** is the script used for preprocessing the dump of Wikipedia,

**train_emeddings.py** is a Python script used to train the CBOW model in order to obtain the word embeddings,

**model.py** is a Python script performing the training and evaluation of the models performance

**preprocess_reviews.sh** is a bash script used to extract the user reviews.

The scripts are well-commented and all (expect **experiment.py**) support **-h** option to provide information on their usage.

## Appendix B

The attached DVD contains the modified dataset in the **data** directory.
For the purpose of our experiment we used only **suffled_anot_first.xml**, but the dataset contains the following files:

**positive_summary_anot_first.xml** containing 1000 short samples of positive user reviews,

**negative_summary_anot_first.xml** containing 1000 short negative examples of user reviews,

**shuffled_anot_first.xml** contains the lines of the two files above, randomly ordered,

**positive_summary_anot_lenght.xml** containing 74 long samples of positive user reviews and

**negative_summary_anot_length.xml** containing 85 long negative examples of user reviews.