

LU TP 14-nn
June 11, 2018

Natural Language Processing in Artificial Neural Networks
Sentence analysis in medical papers

Konstantin-Klemens Lurz

Department of Astronomy and Theoretical Physics, Lund University

Master thesis supervised by Mattias Ohlsson



LUND
UNIVERSITY

Abstract

Convolutional Neural Networks (CNNs) and pre-trained word embeddings have revolutionized the field of Natural Language Processing (NLP) during the last years. In this project, CNNs are used on top of the Word2Vec word representation for a sentence classification task on medical research articles. Both individual networks for each category as well as a combined classification network are optimized and achieve average AUC scores of 0.89 and 0.82 respectively. A comparison with the results of a collaborating group using Support Vector Machines (SVMs) shows that simple CNNs can now compete with SVMs in this formerly SVM dominated area. In an extension, Recurrent Neural Networks (RNNs) are also trained on the same task and shown to be unfavorable compared to CNNs because of their lack of stability.

Acknowledgements

First of all I would like to thank the supervisor of my thesis, Mattias Ohlsson, for providing me with a very interesting and promising topic and for his guidance and advice on all issues related to it. I would also like to thank Johan Frid and Jonas Björk for the fruitful collaboration in our projects and for providing the data that was used in this project. Further, I would like to thank the department of astronomy and theoretical physics and all the people working in it for providing me with an office and access to high speed GPU computers during a period of more than one year. Finally I want to give a big thank you to all my friends at Fysicum for their mental support at the countless early breakfasts, unhealthy fikas and late night Rydberg pubs.

Popular science summary

When Alan Turing stated his world famous "Turing Test" in 1950 he made clear that for a machine to be considered intelligent it must, among other skills, have conversational capacities comparable to a human being. Since then, scientists in the field of Natural Language Processing (NLP) have undertaken uncountable attempts to come closer to this objective. While we are still far from reaching human-like language skills of machines, the field of Artificial Neural Networks (ANNs) has contributed very promising improvements in the past years.

ANNs mimic the functioning of the human brain by passing signals from node to node (the artificial neurons) via weighted connections (the artificial synapses). The connection weights are flexible and can be tweaked and twisted just like synapses in the brain can grow stronger or weaker. The effect is that weights and synapses alike can adapt to a problem and finally solve it, a process which is known to everyone as "learning". Apart from a strong increase of computational power, the substantial progress of the field in the last years is mainly related to new architectures of how the ANNs are built and connected. For instance, the so called Convolutional Neural Network (CNN) was inspired by the human visual cortex and is set up in a way that it can detect shapes and objects in a picture independently of the objects location and scale. Even though this may sound rather trivial to a human observer it is a great challenge for a computer. While it was initially used for image recognition tasks, scientists have now come to appreciate its value in NLP as well. Just like objects in pictures, words and phrases can also appear in different locations in a sentence: "I will eat today" and "Today I will eat" have the same meaning (except for the emphasis) but look very different to a computer. The ability to recognize locations within the sentence also creates a concept of context and thus memory, essential parts of understanding language.

Before language can be handled by machines, its most basic building blocks, the words, need to be transformed into something the machine can process, namely numbers. In 2013 researchers made a groundbreaking discovery when they fed the entire content of Wikipedia into a powerful ANN and trained it to create its own language which they called Word2Vec. In this language, every human word is associated with a vector, an array of numbers, in a 300 dimensional vector space. While the 300 dimensions of these word vectors do not correspond to any concepts that are known to humans, the computer seemed to have encoded both the semantic and syntactic meaning of the words. For example, subtracting $vector("Man")$ from $vector("King")$ and adding $vector("Woman")$ to it did indeed result in $vector("Queen")$.

In this project, CNNs are used together with the machine language Word2Vec in order to solve a problem in medical science. Due to the high publication rate of medical papers, the findings of these papers do not reach their application with the common doctors and hospitals any more. An Artificial Intelligence is thus needed to extract and process the contents of the articles and present them to the doctors when consulted about a specific topic. The first step for this purpose is done in this project, the training of an ANN which can classify sentences according to their type of content such as "Methods", "Results" or "Aim". The future step will then be to extract the content which had been located previously by our network.

Contents

1	Introduction	3
1.1	Outline of the task	4
1.2	Related work	4
2	Data	6
2.1	Malmö Diet and Cancer Cohort	6
2.2	Categories and data labeling	6
2.3	Data pre-processing	8
2.3.1	Tokenization and string cleaning	9
2.3.2	Construction of vocabulary look-up table	10
2.3.3	Padding	10
2.4	Model validation	11
2.4.1	Area under ROC curve	11
2.4.2	K-fold cross validation	12
3	Methods	14
3.1	Artificial Neural Networks	14
3.1.1	Convolutional layers	19
3.1.2	Recurrent layers	24
3.1.3	Autoencoders	27
3.2	Regularization techniques	28
3.2.1	Dropout	29
3.2.2	L2-Regularization	29
3.3	Word representations	29
3.3.1	Word2Vec	30
3.3.2	GloVe	33
3.3.3	N-grams with Naive Bayes	34
3.4	Support Vector Machines	34
4	Implementation details	37
4.1	Software	37
4.2	Reproducibility	37
4.3	Training parameters	39
5	Results and discussion	40
5.1	Convolutional networks	41
5.1.1	Individual categorization	42
5.1.2	Combined categorization	47
5.2	Recurrent networks	49
5.3	Word2Vec performance	51
6	Conclusion	52

7 Outlook	53
A Appendices	54
A.1 Comparison of optimization methods	54
A.1.1 Stochastic Gradient Descent	54
A.1.2 AdaGrad optimizer	54
A.1.3 Adam optimizer	55
A.1.4 Result of comparison of optimizers	55
A.2 CNN individual categories	57
A.2.1 Network size	57
A.2.2 Regularization	58
A.2.3 Loss-AUC curves	60
A.3 CNN combined categories	61
A.3.1 Regularization	61
A.4 Software	62
Acronyms	63

1 Introduction

Language is the essential tool that makes communication within our societies possible. The interest of embedding human language into machines is thus little surprisingly as old as the computing age itself and has become a major goal in the area of Artificial Intelligence. The well known Turing test stated already in 1950 that in order for a machine to be considered intelligent, it must have conversational capacities comparable to a human [1]. While current technologies are still far from achieving these abilities, considerable improvement in the field of NLP has been accomplished:

The main focus in NLP was initially devoted to machine translation where the first benchmarks were set by the Georgetown - IBM experiment 1954 [2] on the translation of 60 Russian sentences. The initial euphoria after this experiment, however, was followed by disillusionment when the researchers became aware that successful NLP needed a far deeper and more sophisticated understanding of the linguistic system than anticipated.

The next major breakthrough after years of little improvement came in the late 1980s with the introduction of machine learning. Up to that point, researchers had directly implemented complex grammatical rules by hand into a machine similar as students are taught second languages in most western school systems. The approach of machine learning on the other hand, was to let the machine learn and incorporate the rules itself simply by exposing it to language and giving feedback. This approach is rooted in the way small children learn their mother tongue. The introduction of machine learning together with the increase of computational power made it possible to exploit already existing large multilingual text corpora such as policies from the EU and the Canadian government. These are available in all of the respective institutions official languages and provided valuable data for the machines to train on [3]. The development of the internet in the 1990s then suddenly gave rise to almost infinite supply of training data, the entire content of the World Wide Web. As opposed to the multilingual government policies, however, this data is raw and not annotated. More sophisticated methods to deal with unlabeled data, so called unsupervised learning methods, had to be developed such as the Word2Vec embedding [52]. One specific branch of machine learning, the so called ANNs, started to draw the attention of the science community by setting new benchmarks with more complex network structures such as Convolutional Neural Networks (CNNs) [32] and Recurrent Neural Networks (RNNs) [41]. Further increase of computational power together with algorithmic improvements (Dropout, ReLu and others, see Chapter 3) then marked the beginning of the Deep Learning era around 2012 [1]. This resulted in the current popularity and reputation of all machine learning fields, including NLP, even outside academia and gives reason to be optimistic about what is yet to come.

1.1 Outline of the task

This aim of this project is to classify sentences into categories depicting their type of content. This NLP classification task is done by two research teams in two different approaches: Johan Frid et al. [6] use Support Vector Machines (SVMs) while we tackle the problem with the help of ANNs, more specifically CNNs and RNNs.

The sentences to be classified are extracted from abstracts of medical research articles. The broader idea behind this is the following:

The rate of new scientific publications all over the world is growing exponentially. In the field of medical research specifically, the number of publications doubles every 7.8 years [4]. While the advances in medicine connected to this growth of medical research have been of great benefit to our societies until now, future advances are put at risk by this very same growth. The mere number of publications is so excessive that the valuable information they contain does no longer reach its destination, the common doctors and hospitals. The every day commitments in medical services simply do not allow enough time to read up on all the newly published research.

A possible solution to handling big data in general is the use of machine learning to help structure and process this data and to retrieve the relevant information. In the case at hand, the medical research papers, the aim is for the doctor to consult the program for information on a specific topic or set of patient characteristics and the machine would then search the academic research data base. As a simple example, he or she might ask for the connection between lung cancer and smoking. The ANN would then be expected to answer that in most studies a strong correlation between the two was found.

This task has to be divided in several sub-tasks. The program has to learn where and how it can find which information, how to extract, interpret and compare this information and finally how to summarize and present it to the user. This project aims at solving the first step, the task of where to find which kind of information. For this purpose we train an ANN to classify sentences from medical articles into categories depicting the content of these sentences. There are 14 categories in total such as "The information to be found in this sentence contains the article's - Results/Methods/Aim/etc.". For a list of all categories, see Table 2 in Chapter 2.2.

While this project focuses on only the abstract sentences of medical papers, an extension to analyzing entire articles as a whole is intended for the future. Also, the possible application of a medical NLP classifier on medical reports is eminently promising.

1.2 Related work

This project was done in a collaboration with Johan Frid et al. [6] who approached the sentence classification task with SVMs as opposed to our ANNs. The results of these two approaches are compared in this thesis.

The use of CNNs for classification tasks in NLP, as it was done in this project, is

a very active field of research. A general evaluation on how to treat CNNs and which hyper-parameters have the most influence on their performance is given in [7]. The main article that this project is based on was published by Yoon Kim [5]. He uses a simple yet successful CNN architecture with several filters, max-pooling and a final dense classifier layer. The input words are represented by Word2Vec [52] vectors and can be kept static or adjustable during training. He also experiments with a two-channel architecture where the two channels have static and adjustable word vectors respectively before they are combined for the final classification layer. Despite its small size, the network performs well on various data sets, scoring new benchmarks on 4 out of 7 tasks, which include sentiment analysis and question classification. The paper also emphasizes the need to train good Word2Vec embeddings in NLP tasks. A similar yet more complex structure had previously been proposed by [8] and an extra layer for semantic clustering was added in [9].

The choice of word representation is a crucial part of NLP. There are attempts to use CNNs with raw One-Hot-Vectors as input instead of pre-trained word embeddings [10]. This method is further enhanced in the author's second paper where they add an additional CNN layer to produce what they call a "region embedding" [11]. While their networks perform well on large texts, the convolutional region embedding does not capture enough information on short texts as they are not pre-trained. The authors in [12], [13] and [14] experiment with two parallel inputs to the CNN: A pre-trained word embedding and the relative position of the embedded word to the target word.

Instead of choosing words as the fundamental building blocks of language, some authors use higher or lower structures: In [15] and [16], whole sentences are embedded in a semantically meaningful manner for Information Retrieval. The authors of Word2Vec also propose sentence and document embeddings which they call Sentence2Vec and Doc2Vec [17]. On the other hand, [18] and [19] break the text input down to the character level and use this as input. The results are promising for deep networks and large data sets but still lack performance if these are not supplied.

NLP with a RNN structure such as the Long Short-Term Memory (LSTM) units, which was added to this project as an extension, is an active field of research as well. Tai et al. [20], for example, use Long Short-Term Memory (LSTM) units and extend them to a tree structure in order to capture not only single words but phrases as well. The comparison of performance between CNNs and RNNs, however, is an ongoing research project and still debated. Yin et al. (2017) [21] tested CNNs and RNNs directly on NLP tasks and find that they perform very similarly with slight advantages for the RNNs as long as the task is not essentially a keyphrase recognition task. They note, however, that RNNs are very sensitive to the changes in hyper-parameters, especially with respect to hidden size and batch size. Bai et al. (2018) [22] published a comparison of the two network structures from a broader perspective, general sequence modelling. They find that simple CNNs outperform LSTM units across a diverse range of tasks and data sets, while they show longer effective memory.

2 Data

The data that was used in this research project and the information about it was supplied by our collaborating team, Johan Frid et al. [6]. It consists of 1006 sentences taken from 92 abstracts from medical research papers. These papers were taken from the online depositories Plos One and Pubmed with the constraint that the research conducted in the articles must have been based on the "Malmö Diet and Cancer Cohort". The sentences were then later on labeled with respect to 14 categories by two independent annotators, Johan Frid and Jonas Björk. The categories are not mutually exclusive.

2.1 Malmö Diet and Cancer Cohort

The Malmö Diet and Cancer (MDC) Cohort is a database containing medical information from 28098 men and women who were born 1923 - 1945 (men) and 1923 - 1950 (women). The medical information was collected during the years 1991 - 1996 and contains a self-administered questionnaire, diet assessment during seven consecutive days, basic anthropometric and blood pressure measurements as well as blood sampling [23]. Although the original intention was to study the connection between dietary habits and cancer, the usage of the cohort was soon extended to examine many other interesting correlations. The association between smoking, the cadmium in the blood and cardiovascular diseases are prominent examples [24][25].

Since the Malmö Diet and Cancer Cohort is also an associated member of the European Prospective Investigation into Cancer and Nutrition (EPIC) organized by the WHO [26], it is very easily accessible and was already used for more than 500 research projects. The easy accessibility and frequent previous usage make the cohort ideal for this current project.

2.2 Categories and data labeling

The basic categorization process that is achieved in this project takes a sentence as input and assigns it to categories. The sentences come from abstracts of medical research papers that used the Malmö Diet and Cancer Cohort. The categories, 14 in total, were agreed upon by the annotators and are not mutually exclusive. A sentence can therefore be in no category at all, just in one category or in several categories at the same time. In practice, the maximum number of categories a sentence is assigned to turned out to be seven.

There are four rhetorical categories (aim, methods, results, conclusions) which were chosen based on the *IMRaD* structure of scientific writing. The remaining ten categories are fact-based and concentrate more on concepts that are specifically common in the realm of epidemiological literature. The occurrence of the specific facts in each rhetorical category is shown in Table 1.

Table 1: Number (percentage) of the specific facts that occurred within each rhetorical category. Table and data taken from [6].

Specific fact, n(% ¹)	Rhetorical category				
	Aim	Methods	Results	Conclusions	None
Exposure	85 (93.4)	173 (55.3)	291 (90.1)	150 (91.5)	118 (80.2)
Outcome	83 (91.2)	151 (48.3)	241 (74.6)	138 (84.1)	100 (68.0)
Study design	6 (6.6)	26 (8.3)	4 (1.2)	0 (0.0)	0 (0.0)
Inclusion criteria	3 (3.3)	27 (8.6)	2 (0.6)	0 (0.0)	1 (0.7)
Actual size	6 (6.6)	138 (44.1)	15 (4.6)	1 (0.6)	1 (0.7)
Several cohorts	10 (11.0)	38 (12.1)	15 (4.6)	3 (1.8)	2 (1.4)
Subgroups	18 (19.8)	15 (4.8)	71 (22.0)	37 (22.6)	6 (4.1)
Follow-up period	3 (3.3)	61 (19.4)	5 (1.5)	2 (1.2)	0 (0.0)
Assessment of end-points	0 (0.0)	13 (4.2)	3 (0.9)	0 (0.0)	0 (0.0)
Statistical method	1 (1.1)	66 (21.0)	13 (4.0)	0 (0.0)	1 (0.7)
Total, n(%²)	91 (9.0)	313 (31.1)	323 (32.1)	164 (16.3)	147 (14.6)

The labeling of the sentences was done by two independent annotators, A1 and A2, who used the same guidelines which they had agreed upon before. The difference between the two annotators was:

- A1 is experienced in the field of epidemiology, A2 is not
- A1 received the sentences in the order of the flowing text and was therefore provided with context. A2 annotated the sentences in random order and without context

The annotators then discussed the cases for which their annotations did not match and agreed on a final syncing. In Table 2, a comparison between the two annotators with respect to each category is given. Since the occurrence of some categories is quite low (16 sentences out of a total of 1006 sentences for the least frequent category, Assessment of end-points), the statistical measure of *Cohen's Kappa* is preferred over a simple percentage measurement of agreement. It takes into account the possibility of the agreement occurring by chance:

$$\kappa = \frac{p_0 - p_e}{1 - p_e} \quad (2.1)$$

p_0 = relative observed agreement

p_e = hypothetical probability of chance agreement

¹Percentage calculated within each rhetorical category

²Percentage calculated out of all 1006 sentences in the full corpus

Table 2: Comparison between annotators: number of positives per annotator (A1 and A2), % agreement, Cohen’s Kappa and the final number of positives after syncing. Table and data taken from [6].

<i>Type</i>	A1	A2	% agreement	κ	After Sync
Aim	93	63	96.8	0.778	91
Methods	308	304	96.8	0.925	313
Results	327	317	91.8	0.813	323
Conclusions	159	106	90.9	0.607	164
Exposure	747	801	88.5	0.675	788
Outcome	639	676	87.8	0.730	686
Study design	31	43	95.8	0.411	31
Inclusion criteria	26	27	97.7	0.554	31
Actual size	134	153	96.7	0.866	154
Several Cohorts	53	67	95.8	0.628	62
Subgroups	138	115	89.0	0.499	142
Follow-up period	57	63	98.0	0.823	68
Assessment of end-points	16	42	97.0	0.471	16
Statistical methods	73	66	97.7	0.822	81

Except for "Conclusions", the agreement between the two annotators was high for the rhetorical categories. That is compared to other annotation tasks that mostly score lower on IMRaD, for example [27] with $\kappa = 0.756$. For the other categories, the agreement varies drastically with "Study Design" and "Assessment of end-points" being the weakest. The categories that the two annotators struggled most with to agree on, will later on also be the ones that pose the tougher problems to the artificial neural network. The different quality of ground truth, that the network will be trained on, has to be taken into account when analyzing its performance.

The following sentence that was picked randomly from the corpus shall serve as an example of how the data looks and how the annotations were made. Both annotators agreed on categorizing it in "Methods", "Exposure" and "Actual size":

[The index ranked 17126 participants -LRB- 59 % women -RRB- of the population-based Malmö Diet and Cancer cohort -LRB- Sweden -RRB- on their dietary intakes.]

2.3 Data pre-processing

It is very essential in machine learning to pre-process the input data in such a way that the network can access the important (and the right) information in the input. In natural language processing one usually applies a variety of operations for this purpose. Apart from basic string cleaning and tokenization this often also includes higher order techniques such

as "stemming" and "lemmatisation" which directly address concepts like grammar and structure in a text. Stemming, for instance, transforms words to a root word. It removes common word endings for English words, such as "es", "ed" and "s". For example i.e., "computer" and "computers" become "comput". Lemmatisation transforms a verb to its dictionary base form i.e., "produces" and "produced" become "produce". The reason for doing this is that one does not want the network to treat variations of the same word as completely independent entities. By tracing these variations back to their root and replacing them by it, one makes sure that the semantic information is conserved. This goes with the loss of syntactic information.

This project uses a new and sophisticated word representation in which both semantic and syntactic information can be conserved, see chapter 3.3.1. The higher order techniques mentioned above are thus not applied and data pre-processing is kept to a very basic level described in the next chapter 2.3.1, very similarly to how it was done by Yoon Kim [5]. This, and also the fact that numbers and abbreviations of molecules and medical names are not substituted, is a clear contrast when compared to Johan Frid et al. [6].

2.3.1 Tokenization and string cleaning

Tokenization and string cleaning are two basic operations that make the textual data easier to process for the machine. Tokenization splits the sentences into tokens, where a token can be a word, symbol or number. String cleaning includes a variety of things:

- Set all letters to lower case.
- Substitute all characters except for letters, numbers and ",()'?!" with a space " ".
- Split contractions into two words, e.g. "you've" becomes "you 've".
- Insert spaces in front of all symbols, e.g. "really?" becomes "really ?". Symbols hereby become tokens on their own.
- Reset all double spaces that might have been created to a single space and split tokens by spaces.

Note again that, in contrast to [6], all numbers, ranges and names of genes and molecules are *not* converted into a generic symbol. The hope is here that the word representation will learn by itself that e.g. a four digit number is likely to be a year and that the concept of "year" will also be part of the information stored in the word representation, apart from just the numerical value of the four digits themselves.

The example sentence from chapter 2.2 thus becomes:

*['the', 'index', 'ranked', '17126', 'participants', 'lrb', '59', 'women', 'rrb', 'of',
'the', 'population', 'based', 'malm', 'diet', 'and', 'cancer', 'cohort', 'lrb',
'sweden', 'rrb', 'on', 'their', 'dietary', 'intakes']*

2.3.2 Construction of vocabulary look-up table

All the tokens (words, numbers, symbols) were combined into one vocabulary and a mapping between the token and its index was created. The build up of the vocabulary was done in the order of the occurrence of the token in the corpus. From then on, the token was only referred to by the integer number of its index in the vocabulary. The first index value that was used was "1". The index "0" was reserved for padding, see Chapter 2.3.3. This word representation is called "Bag of Words". In this project, the "Bag of Words" word representation was then later further translated into a higher order word representation called "word2vec", see Chapter 3.3.1.

So far, the example sentence from Chapter 2.3.1 has thus become:

$$\text{array}([3, 122, 1293, 1750, 67, 5, 896, 47, 4, 6, 3, 64, 87, 51, 31, 2, 17, 48, \\ 5, 687, 4, 41, 472, 75, 212])$$

2.3.3 Padding

Padding is an operation that adds a symbol or value to the edges of a vector or matrix in order to increase its size up to a certain desired size. The reason this is done here is that the sentences do not have the same length or number of words. In order to have same sized sentences, the value "0" is padded to the end of the sentences until they all have the same length as the longest sentence in the corpus. The padding value "0" does not stand for any word or symbol and the networks learns that it does not contain any information. In the extension of this project, the network was changed and RNNs were applied instead of CNNs, see chapter 3.1.2. Due to the different functioning of the RNN, the padding had to be done at the beginning of the sentence instead of at the end of the sentence for this type of network. It does not make a conceptual difference though and for CNNs both ways would have been possible. The decision for the CNN was the more intuitive padding at the end of the sentence.

The sentence with maximum length in the corpus has 121 tokens. The example sentences from Chapter 2.3.2 was thus padded to this size and is in the final form:

$$\text{array}([3, 122, 1293, 1750, 67, 5, 896, 47, 4, 6, 3, 64, 87, 51, 31, 2, 17, 48, \\ 5, 687, 4, 41, 472, 75, 212, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \\ 0, \\ 0, \\ 0, 0])$$

2.4 Model validation

In order to make an estimate of the performance of the network, one needs some kind of measure to evaluate how many (and which) of the categorizations the network classified correctly. Due to the specific characteristics of the problem in this project, the Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) was chosen. With this measure of performance the network needs to be tested on an independent validation set to avoid bias. The method that efficiently splits the data corpus into training and validation sets while still providing the network with an optimal input of information is called K-fold cross validation.

2.4.1 Area under ROC curve

The most commonly used performance measure for classification tasks is the accuracy. It simply measures the percentage of correctly classified samples. In the case of the data in this project, however, the accuracy is too rough to adequately describe the performance of the classifier, a more fine-tuned and sophisticated measure is needed. This is due to the nature of the classes which are very unevenly represented by the input data, see Table 2 in Chapter 2.2. The frequency of the classes ranges from 78% down to only 2% of all input sentences. Especially in the case of the rare categories an accuracy of close to 100% could thus easily be achieved by simply classifying all sentences as "negative" with respect to the corresponding rare category. Therefore one needs to look at the true-positive rate (TPR) and the false-positive rate (FPR) individually. The true-positive rate is also known as sensitivity while the false-positive rate is computed as $(1 - \text{specificity})$.

The network itself only outputs probabilities for the input to be in a certain category, see the sigmoid function in Equation 3.2. The experimenter thus has to decide on a threshold probability from where on the categorization is considered positive. While the most natural choice would be 50%, other threshold values can also be reasonable for example if the damage of a false-positive is much more severe than of a false-negative or vice versa.

The choice of threshold value, however, is an additional parameter and influences the estimation of performance. Therefore, one compares TPR and FPR by plotting them against each other in a graph for many different choices of thresholds. The result is called the Receiver Operating Characteristic (ROC) curve and can be seen in Figure 1 for a random example. A threshold independent measure of performance is then the Area Under the Curve (AUC). The term AUC is ambiguous because it could technically refer to the area under any kind of curve. It should therefore be specified as area under the ROC curve, AUROC. However, in the literature the term AUC is used despite its ambiguity and we will do so in the following as well.

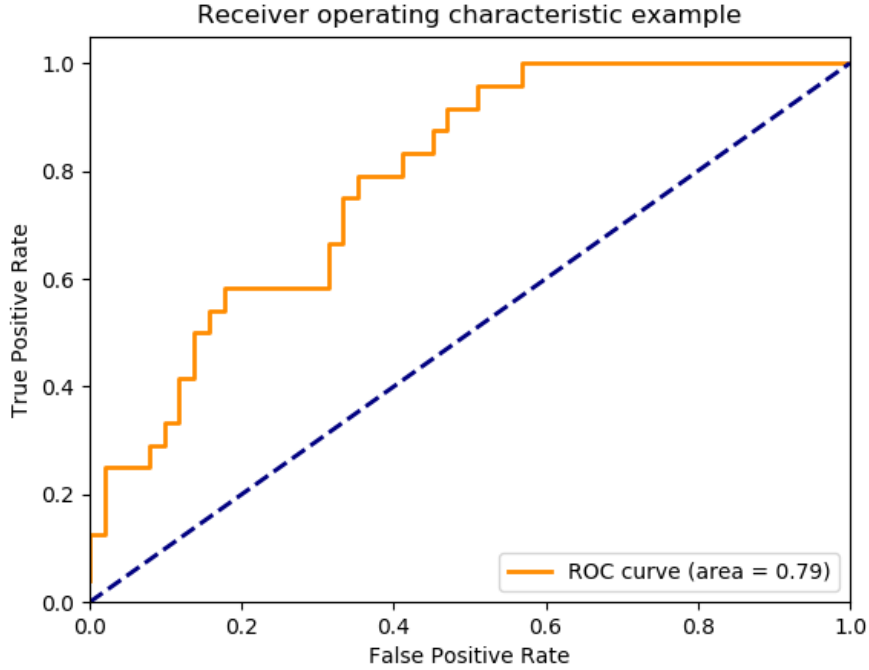


Figure 1: Example for a ROC curve (orange) and the computed Area Under the Curve (AUC). An AUC value of $AUC = 1$ means a perfect categorization performance while $AUC = 0.5$ (blue dashed line) is complete randomness. Source: [68]

2.4.2 K-fold cross validation

The entire data corpus is used for two main tasks: training the network and estimating its performance. For this purpose the data set is split in two parts, the training data and the validation data. They must never be mixed because evaluating the performance on data that the network has been trained on would be extremely biased! The downside of this is that a part of the data, the validation data, will never be used for the improvement of the network and since data is expensive and hard to obtain we would like to put the data we have to the best use possible to train a good network.

The usual approach to tackle this problem is K-fold cross validation. It makes sure that all the data is used by training several networks with different data splits and averaging over it. The exact procedure is explained in the next paragraph. Note however that K-fold cross validation is not used to its full extent in this project due to problems that arise from our specific data set. These problems and their solutions and necessary changes to K-fold cross validation in this project are explained in the end of this chapter.

For a graphical visualization of K-fold cross validation, see Figure 2. The entire data set is split into K parts or "folds". The number of folds, K, is an unfixed parameter that can be chosen taking into account the characteristics of the specific data set. A commonly

used value is $K = 10$. Hereby the data splits are usually done "stratified" which means that the proportions of data samples corresponding to a certain label are kept the same in all folds. One fold is then taken as the test set for estimating the general performance of the network in the end while all the other folds are combined in the so called construction set. This is done for each of the K folds once, so in the end there are K variations of Test-Construction splits. For every variation of this split, the construction data is split into M parts. M again being the number of splits in this construction set. Just as before, M different variations of splits are made each consisting of one validation fold and the other $M-1$ folds combined in the training set. This results in one inner and one outer loop with M and K iterations respectively. In the inner loop, M networks are trained per value of the hyper-parameter that is investigated. The results are averaged and the optimal value for the hyper-parameter is found. Then K networks are tested in the outer loop with this choice of hyper-parameter and again, the results are averaged.

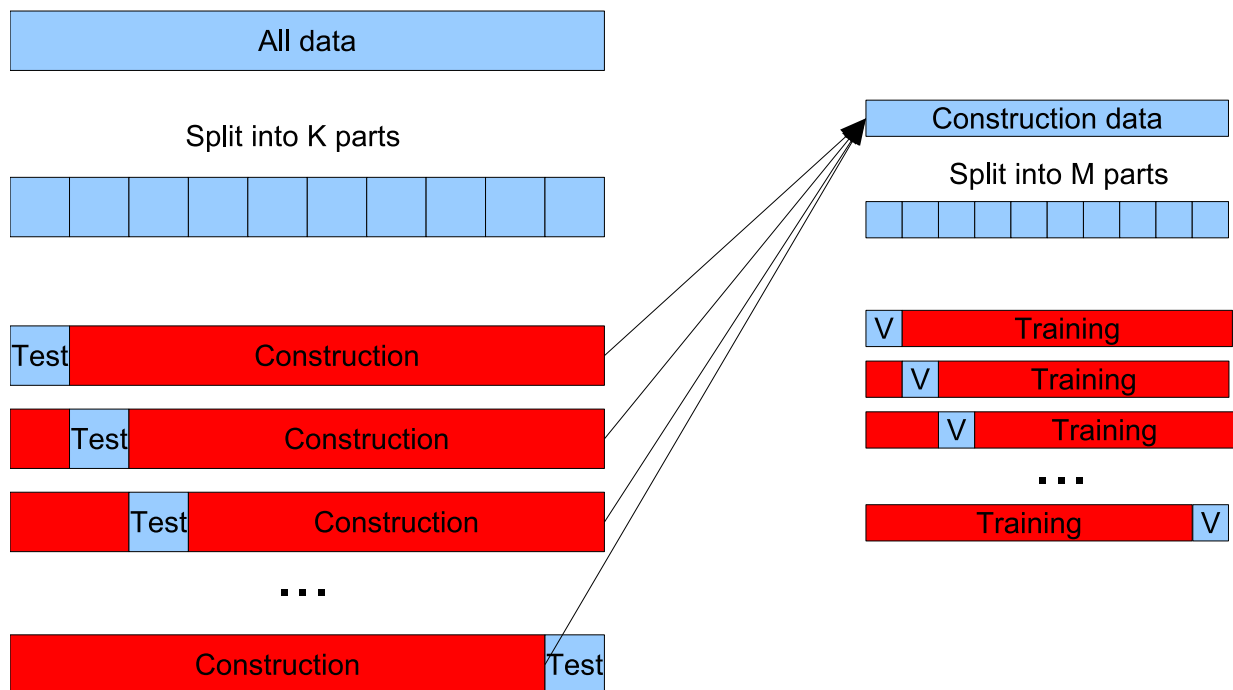


Figure 2: K-fold cross validation: An inner and an outer loop for the search of hyper-parameters and the estimation of general performance. By doing different splits of test-construction/validation-training sets and averaging over them, all the data is put to optimal use.

The specific data set used in this project imposed two problems:

- The frequency of some categories was extremely low, see Table 2. For instance, the least likely category, Assessment of end-points, occurred in only 16 out of 1006 sentences.
- Due to the relative small size of the data set, the performance of the networks depends a lot on the specific split of the K folds. This resulted in a high random fluctuation noise of performance which covered the actual signal.

In order to overcome these problems the following changes to K-fold cross validation were made. Instead of having one inner and one outer loop with many different folds of which each has to contain a reasonable amount of each category, the process was constricted to only one loop. This loop contained $K = 5$ folds instead of the more common choice of $K = 10$. That way, there were still at least three sentences of the least frequent category, Assessment of end-points, in each fold. The hyper-parameter search was done with a seeded split of the folds (and also seeded initializations of weights) in order to combat the high fluctuations and have a clear signal. For the estimation of the generalization performance, $n = 50$ unseeded runs were made and the highly fluctuating results were averaged. This way the generalization performance is technically **not** completely unbiased. However, the bias is trained on a very specific seeded split of training and validation set. Since the performance depends so much on this split and the final performance is the average of 50 variations of these splits, the so obtained performance does also generalize well.

3 Methods

In this section, the methods used to categorize the sentences with machine learning are described. It starts with a short introduction to ANNs as the general learning algorithm of choice in this project. Following the introduction, more sophisticated variations of ANNs are discussed, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and Auto-encoders. After having introduced the general network building blocks, the actual architectures of the networks in this project are presented and techniques to regularize them are explained. Since the representation of words is an essential part in NLP, an overview of the most commonly used word representations is given and a comparison between them and the representation used in this project is made. The chapter is finished with a short summary of SVMs, the machine learning technique that was adopted by the cooperating research team, Johan Frid et al. [6].

3.1 Artificial Neural Networks

An Artificial Neural Networks is a machine learning algorithm that is highly inspired by nature. It is based on how the neurons in the human brain work together to form a powerful information processing apparatus. Simon Haykin defines an ANN in his book *Neural Networks: A Comprehensive Foundation* in the following way [28]:

A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experimental knowledge and making it available for use. It resembles the brain in two aspects:

- *Knowledge is acquired by the network from its environment through a learning process.*
- *Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.*

The human brain consists of approximately 10^{11} neurons which can be seen as its basic computational elements. Each neuron in turn consists of several operative elements which together guarantee the well-functioning of the cell: The cell body receives electrical pulses from other neurons via cable-like structures called dendrites. These electric signals from other neurons add up within the cell body until a certain threshold is reached. Once this is the case, the neuron itself fires an electric signal as well, into a long nerve called the axon. The axon eventually branches off into synapses which are connected to other neurons dendrites with varying synapse strengths. This way, information is transmitted and processed within a network of biological neurons [29].

In an Artificial Neural Network, the neurons are computational nodes. They receive input via weights which mimic the synapse strengths in the biological model. The input is summed up within the node and a bias is added, the equivalent to the threshold in the biological cell. The result is then processed with an activation function and the output is passed on to the next nodes which receive it via weights. A schematic diagram of one computational node can be seen in Figure 3.

The choice of activation function depends on the problem at hand and, to a certain extent, personal preferences. While a linear activation function should only be used for linearly separable problems and networks with only one layer, there is a variety of non-linear activation functions with different advantages and disadvantages. There will be more discussion on activation functions later in this chapter.

The artificial neurons described in Figure 3 can be arranged and connected in many different ways. The most simple way is an architecture that is called Dense Layer where the neurons are embedded in a clear fully connected layer structure. That is, every neuron is connected to exactly and only every neuron of the previous and every neuron of the following layer. Dense layers are feed-forward architectures which means that the flow of information in this network is always in one direction, the forward direction. An example of such a dense layer network can be seen in Figure 4. This example network consists of the obligatory input and output layer, x and y , and two hidden layers in between, h^1 and h^2 . While the size of the input and output layer is determined by the problem, finding an appropriate size and number of hidden layers is one of the main tasks of the researcher. A network that is too small will not have the necessary computational power to solve the problem. A too complex network, on the other hand, takes a long time to train and holds the risk of over-fitting on noise in the data. More on the problem of over-fitting is discussed

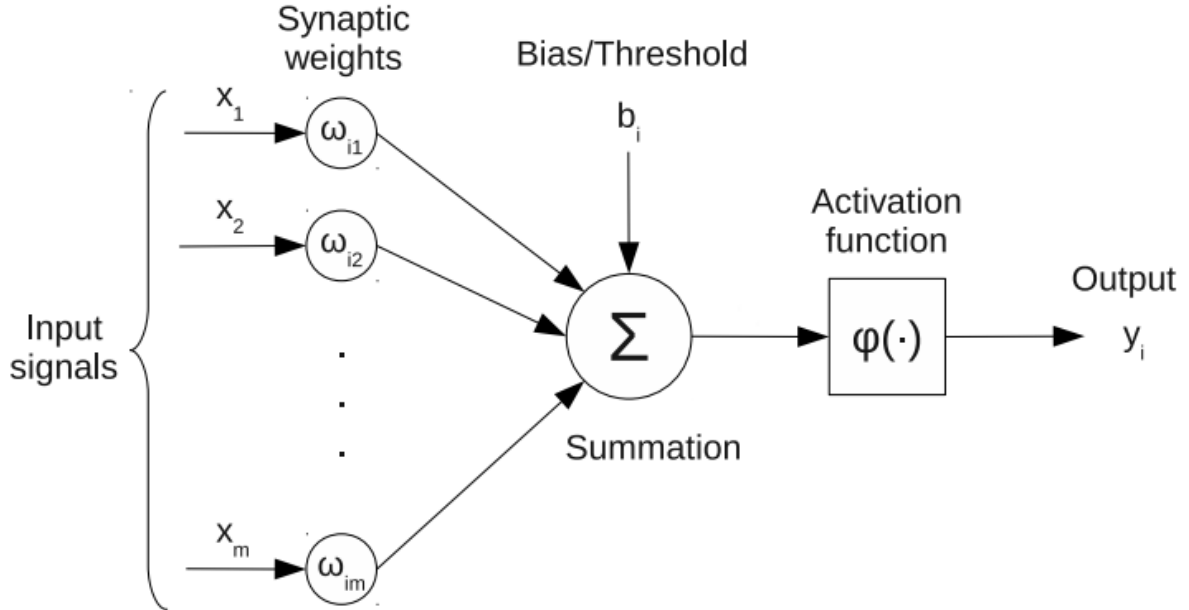


Figure 3: Model of a computational node in an ANN. The input is weighted and summed up, a bias is added and the result is passed into an activation function to produce the output [29].

in Chapter 3.2.

There exist many other (more complex) architectures for ANNs. For instance, one can establish skip-layer connections for example from the input directly to the output layer. This way, the output layer receives both raw input information as well as processed information that went through the hidden layers. Two architectures which are generally used a lot in ANNs and which are also part of this project are Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). In CNNs, the weights between certain neurons are not independent but shared over the network which makes them highly applicable in feature detection. RNNs dissolve the concept of feed-forward information processing and allow feed-backward connections as well. This results in a memory effect within the network and is especially beneficial in problems with sequenced data. CNNs and RNNs will be further discussed in detail in Chapter 3.1.1 and 3.1.2.

From a mathematical point of view an ANN is an approximation function $y(\mathbf{x}; \mathbf{w})$ in dependence of the input \mathbf{x} and with the weight matrix \mathbf{w} as parameter which is optimized on labeled data d_i ¹.

¹There are also techniques without labeled data, called unsupervised learning. Autoencoders like the word2vec representation used in this project are an example, see Chapter 3.1.3 and 3.3.1.

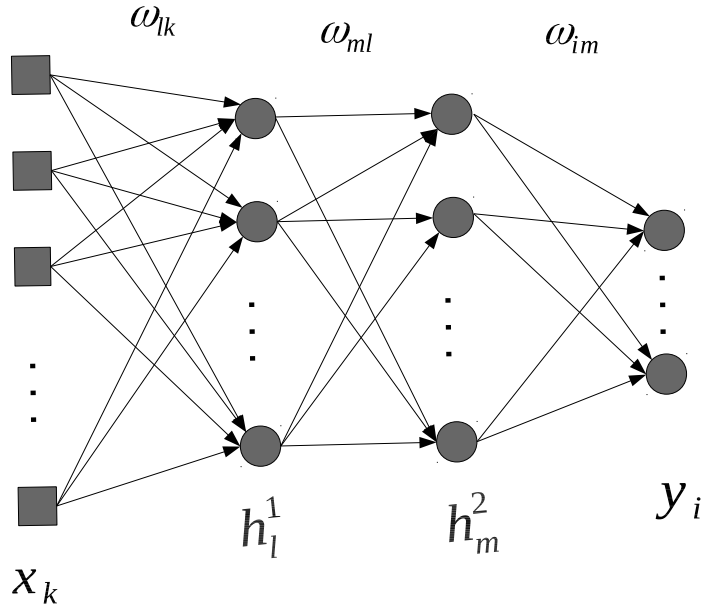


Figure 4: Example of a fully connected dense network with two hidden layers h^1 and h^2 , the input x and the output y connected by the weights w .

A common optimization process is called gradient descent. It involves calculating gradients of an error function with respect to the weights in a method called back-propagation[29]:

The output of each output node y_i is computed by passing the input of the node through its activation function. This project deals with a classification problem with the aim of assigning input sentences to a certain category or class. The output nodes $y_i(n)$ thus represent the probability of the input sentence n to belong to class c_i . Since an interpretation of a probability requires the output to be in the interval $[0, 1]$ and since the classes are **not** mutually exclusive, the output activation function is the sigmoid function, independently of the number of classes¹:

$$y_i(n) = \frac{1}{1 + e^{-(\mathbf{w}_i^T \mathbf{x}(n))}} \quad (3.2)$$

This output $y_i(n)$ is then compared to the true label $d_i(n)$ and an error measure E is computed, the so called cross-entropy loss function. This is done for all N sentences in the data set and the error is accumulated:

$$E = -\frac{1}{N} \sum_n \sum_i d_i(n) \cdot \log[y_i(\mathbf{x}(n); \mathbf{w})] \quad (3.3)$$

In parts of this project, the classification networks will only look at individual classes instead of all the classes simultaneously. In this case the network has only one output node

¹If the classes were mutually exclusive, one would use the softmax function in order to normalize the output probabilities.

and the classification is binary with the simplified binary cross-entropy loss function:

$$E = -\frac{1}{N} \sum_n [d(n) \cdot \log(y(\mathbf{x}(n); \mathbf{w})) + (1 - d(n)) \cdot \log(1 - y(\mathbf{x}(n); \mathbf{w}))] \quad (3.4)$$

The error E is then minimized with respect to the weights \mathbf{w} and the weight updates are obtained by the computed gradient times the learning rate η which is set by the experimenter:

$$\Delta w_{ij} = \eta \cdot \frac{\partial E}{\partial w_{ij}} \quad (3.5)$$

There are many variations and extensions to this basic overview of gradient descent. In this project three different implementations are tested and compared: ADAM, SGD and ADA-GRAD, see the appendix A.1. Each of them is used with the extension of L2-regularization, see Chapter 3.2.2. Note also that the process described above updates the weights after all N input sentences $\mathbf{x}(n)$ have been presented to the network which is called batch learning. Since this type of weight update is accumulated and averaged, it is very stable and robust. The drawback, however, is the relatively low learning speed due to only one update per batch. The opposite of batch learning is called online learning where the weights are updated after every single input of the batch. This technique is fast but highly stochastic and unstable which is why it is rarely used. In practice one usually settles in between these two extreme versions and compromises on so called mini-batches. The data of a batch is split into mini-batches and the back-propagation process and weight update are performed on the average of each mini-batch instead. This is called Stochastic Gradient Descent (SGD). The size of the mini-batches is a free parameter and should be chosen according to the data in the specific problem. In this project, splitting the data batch of 1006 sentences into mini-batches of 20 sentences has been found to be a good choice.

As one can see in Equation 3.5, the calculation of the weight updates involves computing the gradients of the activation functions of all the nodes in the network. This gives rise to the so called vanishing or exploding gradient problem for deep networks. If the gradient of the activation function becomes very small in one layer, the weight updates of all previous layers become small as well since they are multiplied by it. The network gets stuck and does not improve much during learning even though some weights might still need adjustment. In the case of exploding gradients, the modification of weights is not fine enough and the error minimum cannot be found. This is why the choice of activation function has to be made carefully for all the hidden layers.¹ The most common choices for activation functions are the Rectified Linear unit (ReLU), the sigmoid function and the hyperbolic tangent (tanh). Plots for these functions can be seen in Figure 5. Nowadays the ReLU is preferred by most researchers for its ability to keep its gradient non-vanishing, even for large $|x|$. It is proven that ANNs using this activation function are still universal approximators despite the fact that the ReLU is unbound [30]. For this reason it is used

¹For the output layer the activation function is determined by the problem, in this case mutually non-exclusive categorization.

in this project for the convolutional and recurrent layers. A comparison of performance between the sigmoid function and the hyperbolic tangent was made by Karlik et al. [31] with the result that the hyperbolic tangent is preferable in almost all cases. It was chosen in this project for the final hidden dense layer in all respective network architectures. The sigmoid was only used in the output layer.

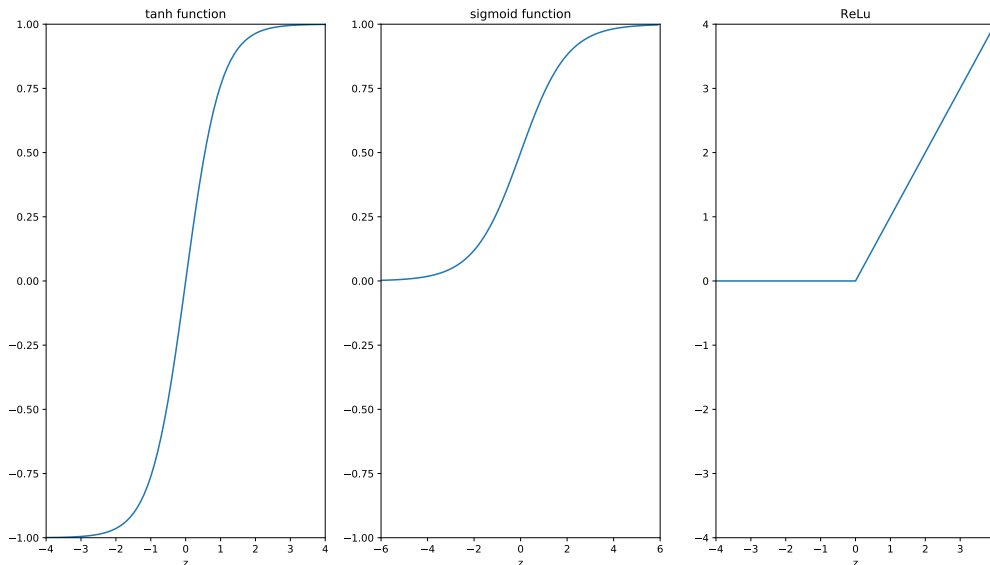


Figure 5: The three activation functions used in this project

3.1.1 Convolutional layers

Convolutional Neural Networks (CNNs) are one of the most successful network architectures in state-of-the-art ANN algorithms. They are inspired by the animal visual cortex and were first applied on hand-written digits by Yann LeCun [32]. Even though their greatest successes still lay in the field of image recognition, they have recently also made their way into Natural Language Processing with impressive results¹. Since CNNs were first developed in image recognition and since their functioning is easier to understand and visualize on the example of images, this introduction will be given in this field. The transition to the field of NLP is then an easy step.

Images are two dimensional matrices of pixel values with a third dimension of color if the image is not black and white. The color dimension, however, is not a dimension that the convolution is applied over. Instead, the operation is done over each of the three color pixel matrices (red, green, blue) separately and the results are added. The convolution is

¹See Chapter 1.2 for an overview of applications of CNNs in NLP and the benchmarks that were set.

thus 2D over the input pixel matrix as shown in Figure 6: A kernel or filter (yellow) slides over the input matrix (red, green or blue respectively) creating an entry in the activation map (orange) for each window in the input matrix. Hereby are the weights of the filter multiplied with the values in the window of the input matrix and the results are added up. The weights in the filter are, of course, subject to the learning process of the network and are shared over all windows of the convolution operation. The sliding stride and the size of the filter are free parameters that can be chosen by the experimenter.² Zero padding can also be added at the edges of the image. The size of the activation map O is then given by [34]

$$O = \frac{W - K + 2P}{S} + 1 \quad (3.6)$$

where W is the height/length of the input matrix, K is the height/length of the filter, P is the padding and S is the stride. The values in the activation map are then passed through an activation function and serve as output towards the next layer in the network.

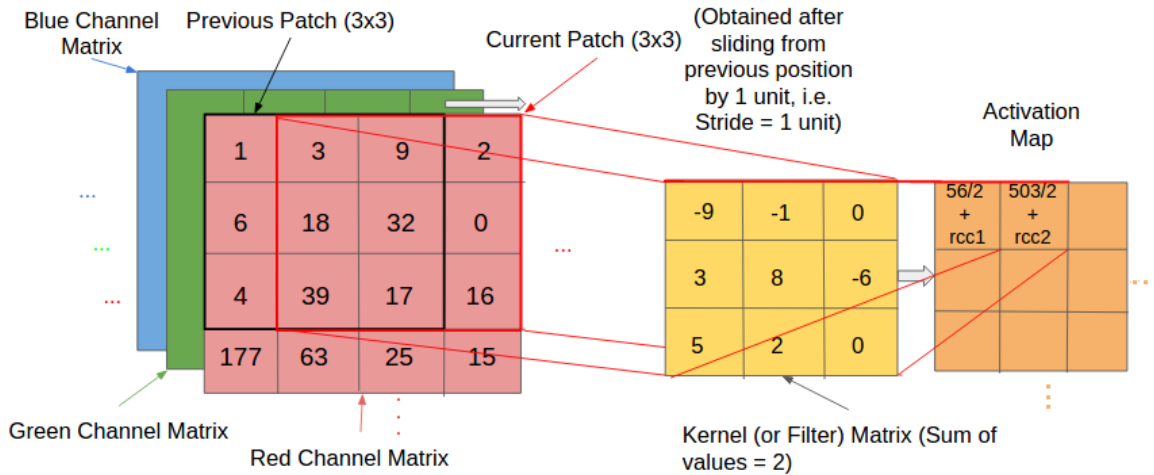


Figure 6: Graphical example of a convolutional operation: The filter (yellow) slides with $stride = 1$ over the pixel matrix of the input image for each of the three colors (red, green, blue). By multiplying the filter values with the pixel values, an activation map (orange) is produced whose entries will later on go through an activation function and serve as output of the convolutional layer. Note that the entries in the activation map are normalized with the sum of the filter values. The term rcc refers to the contribution of the remainder channels. Source: [33]

In CNN architectures, a convolution layer is usually followed by a so called pooling layer. Pooling layers sub-sample their input and can be applied over the whole matrix (global pooling) or over windows. In the example in Figure 7 a 2×2 max-pooling filter is applied with stride 2. There are two main reasons for applying a pooling layer. First, it

²In this example stride $S = 1$ and filter size $K = 3$

can provide a fixed output size which is very useful if a dense categorization layer follows next in the network. For this purpose one can either do max pooling over the complete matrix (global) or with varying window sizes proportional to the respective input sizes. It thus makes varying input image sizes and filter sizes easy to handle. When applied to NLP, this effect is particularly helpful because the input consists of sentences which naturally differ a lot in length. Second and even more important, it significantly reduces the output dimensionality without losing much information. The highest and therefore most important score is kept and the less important lower scores are discarded.

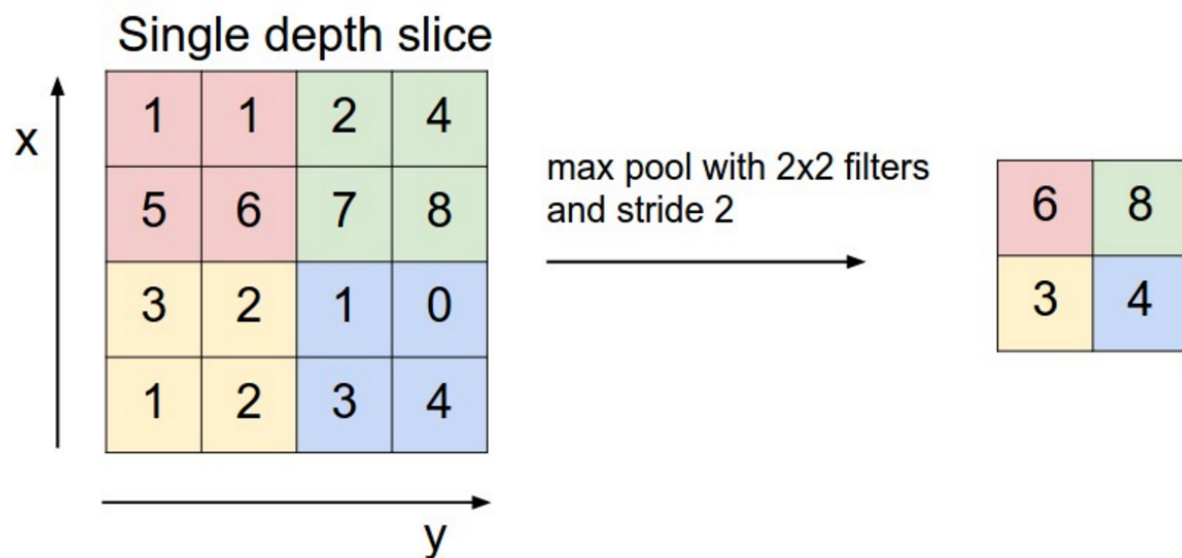


Figure 7: Example for a pooling filter: A 2×2 max pooling filter slides over the input with stride 2. In each window, the maximum value is selected, the other information is discarded. Source: [35]

All in all, a typical image recognition network that categorizes the input images into classes of what is to be seen on the image could look like in Figure 8. The input image is passed into a convolution layer with three different filters. Each of the filters captures a characteristic feature of the images. Since it is the first layer and the first step in processing the input image, this feature is likely to still be very basic. It could for example detect round shapes or edges. The complexity at this level is not high yet and the researchers decided to restrain the size of this layer to (only) three filters. After a pooling operation, the next layer of convolutions is applied. The filters in this level take as input the pooled output of the basic feature detectors and combine the information to detect more sophisticated forms and shapes. The higher complexity of these features is taken into account by expanding this layer to six filters, doubling the first convolutional layer. The output is passed through max-pooling again and is then categorized by two dense layers. For a detailed description and visualization of what the filters in a CNN capture and how they work, see [38] and

[37]. There is also a collection of the most influential work that was done in the field of visualizing CNNs in [39].

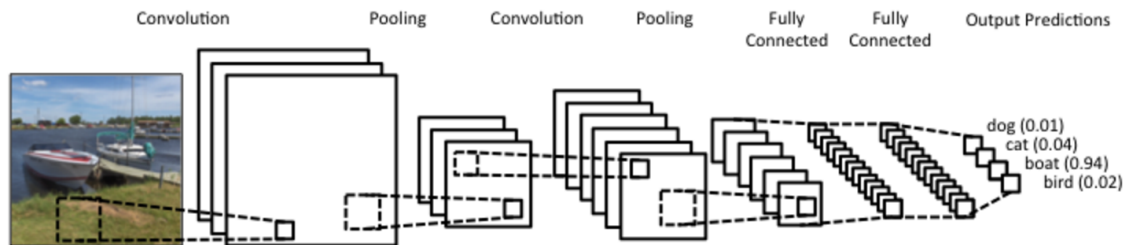


Figure 8: A typical image recognition CNN with two convolutional layers, each followed by a layer of pooling. The categorization is done by the two fully connected dense layers at the end of the network. Source: [36]

The great advantages of CNNs that made them so successful in image recognition are mainly the following [34]:

- The filters in a CNN use shared weights which reduces the number of trainable parameters drastically. The training of the network becomes faster and less computationally expensive.
- The shared weights force the filters to learn from across the entire image. This is a naturally inbuilt and very powerful regularization technique to prevent the network from so called over-fitting, see Chapter 3.2.
- The concept of pooling allows to get rid of much of the unnecessary information and only keeps the essential data, e.g. certain structures and their locations in the image. This again reduces the number of trainable parameters.
- By having filters with a confined size, CNNs implement so called *local receptive fields*. This means that a pixel in an image is mainly put in perspective with its local surrounding. Contrarily, in a simple dense layer architecture all pixels would be treated on the same footing and the spatial information would be lost. Creating a network that can take into account the spatial structure of the input data was the main reason behind the invention of CNNs.
- CNNs cope well with the translation invariance of images. A certain object in an image can appear in different locations, angles and sizes and is still the same object. While this imposes no big challenge to a human observer, ANNs have struggled with it for a long time. The concept of sliding the filters over the image deals with translation invariance very efficiently.

By looking at the advantages and strengths of CNNs one can easily see why their application was extended from images to NLP. The reduction of trainable parameters, the ability to be trained fast, and the inbuilt prevention of over-fitting are welcome characteristics in any machine learning related problem. However, the use of local receptive fields and the ability to handle translation invariance are of significant importance when analyzing human language. Words in a sentence are usually semantically and syntactically connected to their close surrounding words, rather than to a word at a position further away in the sentence. Take for example the sentence: *"I went to the old supermarket this morning"*. The words "I" and "went" are clearly related, as are the words "old" and "supermarket" or "this" and "morning". The concept of looking at the words with respect to their local environment is fulfilled by the local receptive fields in a CNN. However, the example sentence could also be formulated in the following way without changing its meaning (except for a different emphasis): *"This morning I went to the old supermarket"*. Just like images, sentences are translational invariant. As described above, CNNs are capable of processing this spacial information.

When using convolutional layers for NLP, one has to make some small adaptations of the CNN compared to image recognition. As described above, a convolution operation over an image is performed over **two dimensions**, its height and width. In NLP the input to the network is a sentence which consists of words. In this case, the filters slide over only **one dimension**, namely the length of the sentence. The words in the sentence are transformed into a vector representation¹ called *Word2Vec* which is described in Chapter 3.3.1. So a sentence is still presented as a two dimensional input to the network, just like an image, with the dimensions "length of the sentence" and "length of the word vector representation". However, since there is no spatial information in the word vectors, a convolution over this dimension does not make sense. Therefore, the filters have a fixed width of the size of the word vectors and slide solely over the dimension of the sentence length. A diagram of such a CNN applied in NLP can be seen in Figure 9. It is the network architecture that was used by Yoon Kim in his work about CNNs for sentence classification [5] and is one of the main influences that this project is based on.

¹This project experiments with word vectors of length 60 and 300.

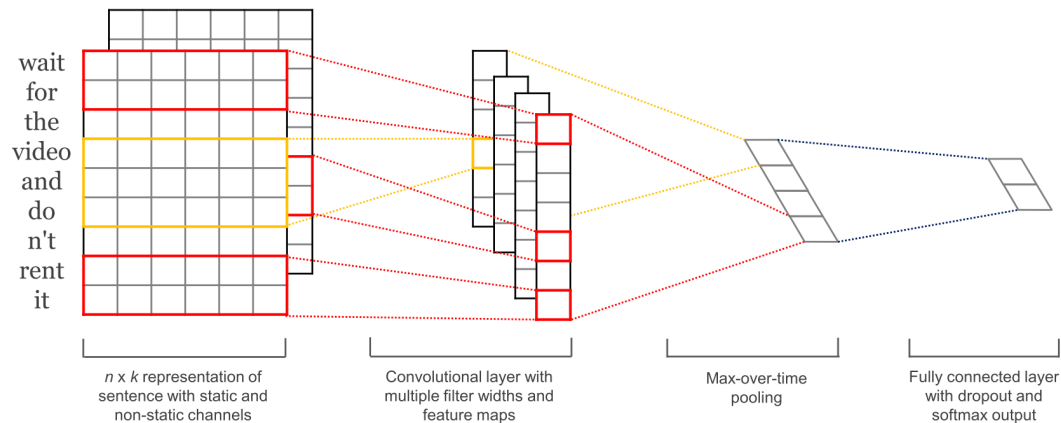


Figure 9: Yoon Kim's CNN for sentence classification. The second input matrix in the background is part of a two channel method and is not of importance here. Source: [5]

3.1.2 Recurrent layers

Another successful network architecture for NLP problems is the so called Recurrent Neural Networks (RNNs)¹, first developed by Jeffrey L. Elman [40]. It is mainly used in machine learning when handling sequenced data or time series. While CNNs are an effective tool to capture information over spatial dimensions, RNNs are capable to do the same over the dimension of time. When using RNNs, the sentences in the input data are thus not interpreted as spatial objects of length "number of words" as in the case of CNNs but as a time sequence of the words they consist of. This is arguably the more natural and intuitive way to look at human language. The approach with RNNs was added to this project as an extension.

The main innovation in RNNs compared to ordinary feed-forward ANNs is that feed-backward connections are accepted as well. This allows previous input to stay in the network and to be coupled with the current input. The result is a memory effect which is essential to NLP because a single word can often only be understood to its full extend if the context that it appears in is known. The introduction of feedback connections complicates the concept of back-propagation remarkably because every feedback connection effectively expands the network by a whole new layer per time step, see Figure 10. This makes the back-propagation process very lengthy and computationally expensive. Also, as in all deep networks, vanishing or exploding gradients can become an issue. One solution to this problem is to truncate the feedback connection after a certain number of time steps, effectively shortening the time range of the memory effect. In 1997, Juergen Schmidhuber et al. introduced another solution to deal with long time series, namely the LSTM units [41].

¹See Chapter 1 for an overview of the applications of RNNs in NLP and the benchmarks that were set.

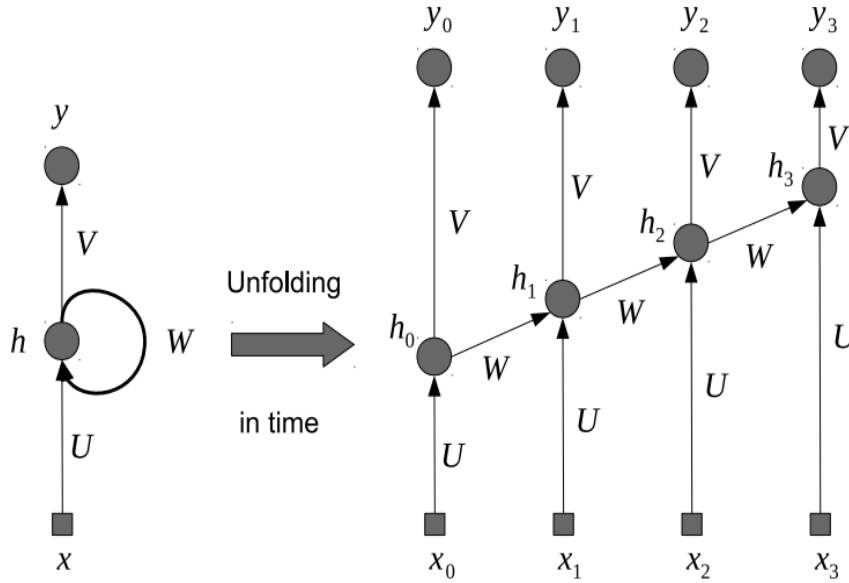


Figure 10: A network with a simple feedback connection (left) can effectively be turned into a feed-forward network by adding a new layer to the network for every time step. This is called unfolding in time. The weights U , V and W in these layers are shared and the inputs x_n are direct inputs to only their respective layer, effectively implementing skip-layer connections. Source: [29]

The basic functioning of LSTM units is as follows [29] [42], illustrated in Figure 11. The main idea behind LSTM units is to introduce an additional variable, the so called cell state C_{t-1} . This cell state is the memory collected from all the previous time steps and is passed on through the network. Each time step it is altered in the units with the use of structures called gates. It is shown as the horizontal line in the top of Figure 11. It serves as an additional input to the unit, besides the output of the previous time step h_{t-1} and the new input x_t .

The first decision that has to be done by the unit is how much of the cell state C_{t-1} to keep and which information to discard. This is done by the forget gate f_t which takes the previous time step output h_{t-1} and the current input x_t as input and passes it through a sigmoid function $\sigma(z)$. The output of the sigmoid is then a number between $[0, 1]$ which is multiplied with C_{t-1} , effectively filtering out parts of C_{t-1} . The information from the output from the previous time step plus the current input is thus used to decide how much of the information in the memory to keep. In the example of sentences as time series, the previous unit might have looked at a noun and its cell state C_{t-1} might contain the gender of this noun, among other things. The current unit is looking at a noun again x_t , would then identify the gender information about the previous noun h_{t-1} and decide to drop it because it would only need this information if it looked for example at an adjective. Later on it would then add its own gender information to the cell state and pass this on.

The next decision is about what and how much to add to the cell state. This is done in two parts: A candidate for the new cell state c_t is created by passing the old output h_{t-1} and the current input x_t through a hyperbolic tangent $\tanh(z)$ to create values between $[-1, 1]$. Simultaneously the same input is used to decide how much of the candidate cell state to actually update. This is done with a sigmoid function, called the input gate i_t . The new information is then added to the old cell state to create the new cell state C_t . One can choose the forget gate f_t and the input gate i_t to share their weights so that one type of information is only replaced by the same kind. Alternatively, they can be independent which makes it possible to add and delete whole categories of information. In the example from above, this is the step where the new gender is selected by i_t and added to the cell state.

Finally, the unit needs to decide on what output h_t to generate. For this purpose it uses the updated state of the cell C_t , pushes it through a hyperbolic tangent function to get values in $[-1, 1]$ and multiplies the result the output gate o_t . The output gate works in the same way as f_t and i_t to select relevant information with a sigmoid function. Both the cell output h_t and the cell state C_t are then passed on to the next cell while h_t is also an output of the whole network.

All these decisions made by the cells which were described above are subject to the learning process. They depend on weights which are adjusted during training. The mathematical expressions for the gates and outputs are thus the following:

$$\begin{aligned}
 f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
 i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
 o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
 c_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\
 C_t &= f_t \cdot C_{t-1} + i_t \cdot c_t \\
 h_t &= o_t \cdot \tanh(C_t)
 \end{aligned}$$

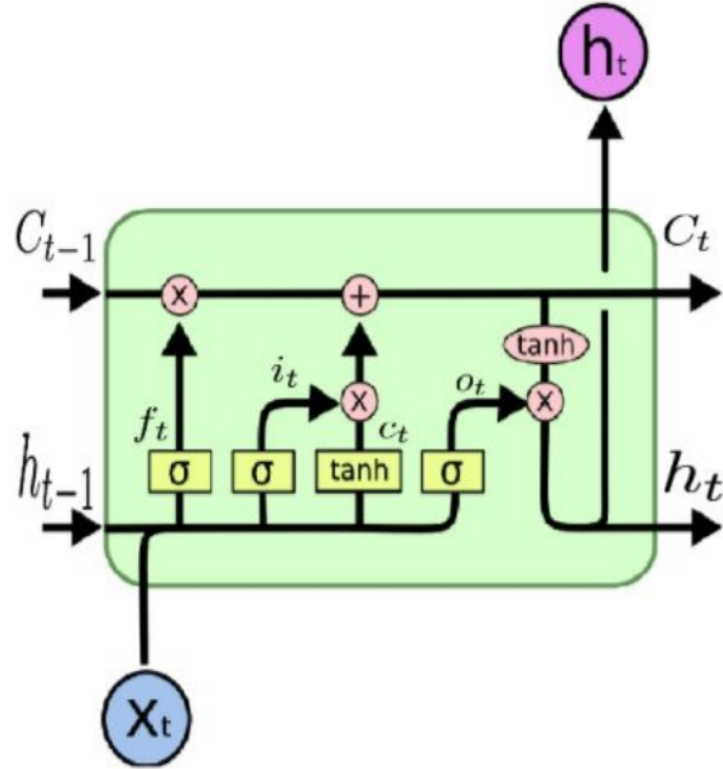


Figure 11: Schematic diagram of a LSTM unit. The raw input x_t , the output of the previous time step h_{t-1} and the previous cell state (memory) C_{t-1} serve as input to the unit. It then processes the information with the forget, input and output gate (f_t , i_t , o_t) and obtains the new cell state C_t and cell output h_t . Source: [42]

3.1.3 Autoencoders

Autoencoders are a type of ANN that does not require labeled data. Instead, they map the input onto itself with the aim of finding a lower dimensional, compressed encoding of the input data. They are therefore often used to pre-process data before it is fed into the actual ANN.

Autoencoders are fully connected feed-forward networks with an uneven number of hidden layers. They are symmetric around the middle layer with the restriction of the middle layer being smaller than the input/output layer. This forces the network to find a lower dimensional representation of the input data while the data flows through the central bottleneck. The result is a structure that resembles a butterfly, see Figure 12. The P dimensional input is mapped onto itself with a M dimensional hidden layer ($M < P$) in between. The activation functions can be linear or non-linear. The network is then trained to produce output that resembles the input as accurately as possible. The error function is consequently [29]:

$$E(\boldsymbol{\omega}) = \sum_i^M (y_i(\boldsymbol{x}, \boldsymbol{\omega}) - x_i)^2 \quad (3.7)$$

The number of hidden nodes M is then the number of dimensions the new data representation has and the trained weights $\boldsymbol{\omega}$ are the new lower dimensional representation. In this project, an autoencoder is used to produce the word representation word2vec, see Chapter 3.3.1.

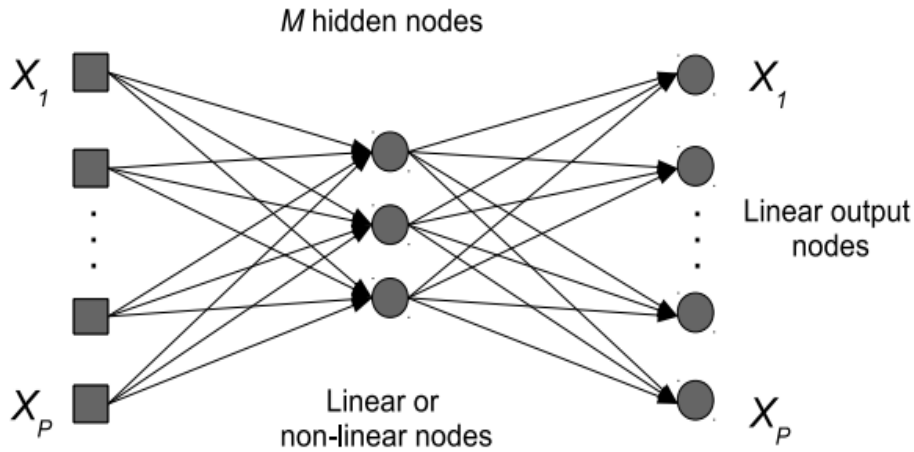


Figure 12: A simple autoencoder with one hidden layer. The input x_p is mapped onto itself with the constraint of having to go through a lower dimensional layer. In the optimal case the output y_p is thus close to identical to x_p . This results in a dimensionality reduction with the new representation of the data having M dimensions, $M < P$. Source: [29]

3.2 Regularization techniques

One of the main challenges when training an ANN is to prevent it from so called overtraining. Overtraining occurs when the network starts to train on the noise that is present in the data rather than the signal. It results in an effective reduction of the training error but yields poor generalization on the test set.

The main reason for overtraining is an unduly high complexity in the network which provides too many degrees of freedom. These excessive degrees of freedom make the network adapt too finely to the training data and the ANN function starts to depict the noisy fluctuations on top of the signal as well as the signal itself. An intuitive approach to tackle this problem is to reduce the level of complexity in the network in order to dispose of the surplus of degrees of freedom. However, a certain amount of complexity is needed in order to solve the problem at hand and the researcher will need to find a compromise when choosing the size of the network.

There are various other techniques to tackle over-fitting by regularizing it which can be

embedded directly in the network and/or optimization process. These methods allow the use of a large and powerful network architecture while still preventing it from fitting noise. This project uses the regularization techniques of Dropout and L2-Regularization.

3.2.1 Dropout

Dropout was proposed by Srivastava et al. [43] as a very efficient method to avoid the common problem of over-fitting, especially in deep networks. The basic idea behind it is to tackle over-adaptation by randomly dropping individual nodes with their connections during each epoch in the training process. The nodes can thus not "rely" on each other in a sense that they have to take over the tasks that were previously done by the dropped out nodes. This prevents them from being able to highly specialize and gives major improvements in the networks generalization performance.

Effectively, Dropout results in the training of large numbers of thinned out networks. During the test phase, however, Dropout is not applied and the thin networks are combined to the original large network which averages over the thin networks. The number of connections thus increases during this phase but the weights of these connections were trained based on only a small number of connections. This can be accounted for by re-scaling the weights with the dropout rate to make sure that the overall average input to each node stays constant.

3.2.2 L2-Regularization

L2-regularization adds a penalty term to the error function. This term scales with the size of the weights and thus forces the network to keep them small [29]:

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2} \sum_i w_i^2 \quad (3.8)$$

Here, λ is a free parameter that has to be tuned by the experimenter to set the necessary amount of regularization. Some of the networks in this projects have layers with a rather small size, especially the networks that focus on only one category, see Chapter 5.1.1. For these networks, L2-regularization is favorable over Dropout because dropping out nodes in an already thin layer can lead to complications in the optimization process.

3.3 Word representations

In Natural Language Processing, the input to the machine learning algorithm consists of letters, words, sentences or documents. These concepts do not exist for computers and they need to be mapped to a representation that can be numerically processed. Since the fundamental input elements in this project are words, the state-of-the-art word representations, also called word embeddings, shall be discussed and compared in this chapter.

The most trivial word representation is the **One-Hot-Vector**: After pre-processing the words, see Chapter 2.3, all the words in the corpus are listed in a vocabulary and each word is assigned its index in this vocabulary. The index in the vocabulary is random and does not bear any information, usually one chooses the order of where the words occur in the corpus for the first time. This is the status in the end of Chapter 2.3. Every word is then assigned a vector of the length of the total vocabulary size V with all entries 0 and only one entry 1, at the word index.

More sophisticated methods aim to capture the semantic and syntactic relationships between words in their vector embeddings such that the vectors of two related words are close in terms of cosine similarity. They do so by using unsupervised learning algorithms which means that they have no need for labeled data. All these methods have in common that they deduce the characteristics of a word solely from its context, an idea that was proposed by the English linguist and language philosopher John Rupert Firth in 1957 [51]: *"You shall know a word by the company it keeps"*.

There are two main approaches to exploit a word's context in order to describe and define it: The global and the local approach. Latent Semantic Analysis (LSA) is the most known embedding technique that uses global context information, namely the occurrence of the words in different documents. The arguably most used model that uses local context is the Word2Vec embedding which was developed by Mikolov et al. at Google in 2013 [52]. It takes as context a window of words that surround the target word. The different types of information gathered from these two approaches lead to different strengths of each model: The global method finds semantic relatedness as in "Boat" and "Water" whereas the local method is better at capturing semantic similarity, like "Boat" and "Ship" [53]. The Global Vector (GloVe) [54] embedding aims to unify the strengths of both approaches.

The word embedding model that was ultimately used in this project is the Word2Vec model. GloVe, however, incorporates interesting features as well and could be a future extension for the project.

3.3.1 Word2Vec

The Word2Vec model by Mikolov et al. [52] uses the local context of a word as an input for the mapping onto the word embedding. Fundamentally, it works much like an auto-encoder, described in Chapter 3.1.3. However, while a classic auto-encoder maps the input onto itself with the aim of conducting a Principal Component Analysis in between, the Word2Vec network relates a word and its surroundings. There are two variations of it: In the Continuous Bag-Of-Words (CBOW) version, a window of n words before and n words after the target word are given as input and the network aims to predict the target word. The skip-gram model, on the other hand, tries to predict the surrounding $2 \cdot n$ words with the central target word as input. For a graphical illustration, see Figure 13.

Note that the word representations for the words that are fed into the Word2Vec network are simple One-Hot-Vectors. After successful training, the new Word2Vec embedding con-

sists then simply of the trained weights of the network. The dimension of these new word vectors is thus the size of the projection layer. The authors argue that both variations yield very similar results with skip-gram performing slightly better on large corpora while being computationally more expansive. Since the corpus in this project is not large (29306 words compared to 1.6 billion in Mikolov et al.) the difference in performance is insignificant and the faster CBOW model was used.

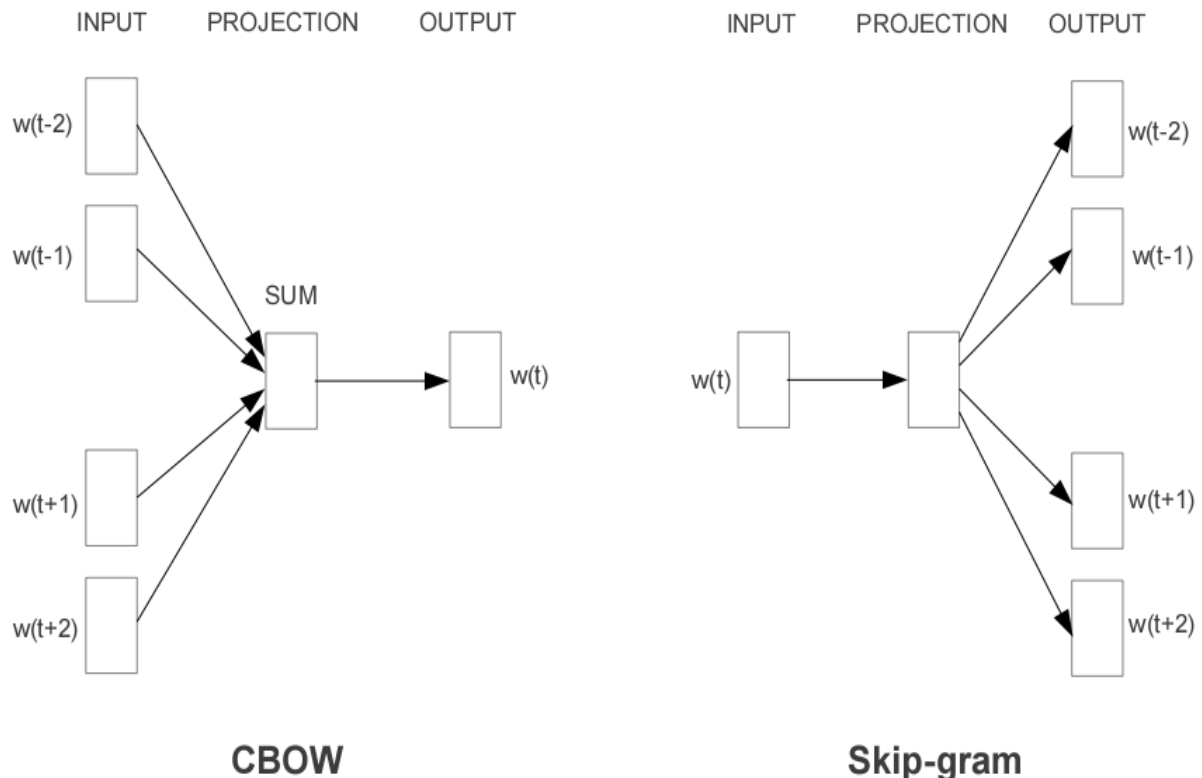


Figure 13: The Continuous Bag-Of-Words (CBOW) and Skip-gram networks in an example network with word context window $n = 2$. While the CBOW model predicts a word based on its surroundings, the Skip-gram model does the opposite by predicting the surroundings based on the central word. The Word2Vec embeddings are then the weights of the trained network. Source: [52]

There are a number of additional pre-processing methods released by the authors in a second publication that improve the performance of Word2Vec [55]:

- Down-sampling of common words. Very frequent words like "the" or "and" do not contain a lot of information as they show up in so many different surroundings that their characteristics can not be specified by the network. In fact, they dilute the information for the other words when they appear in their context windows. The range for down-sampling of these words that is recommended in the Gensim [65]

Word2Vec implementation is $[0, e^{-5}]$. Since the corpus in this project is not very large, a down-sampling of e^{-3} was used.

- Deleting rare words: It is difficult to learn meaningful word representations for rare words because they do not contribute enough input data for the network. However, deleting rare words in this project would have resulted in a drastic reduction of the text corpus which consists of an above-average amount of rare words (names of molecules, medical terms, abbreviations, etc.) and was thus not applied.
- Dynamical windows size: A dynamical windows size enables the network to learn from different context ranges. The advantage of this is intuitively understandable as words further away from the target word should be less/differently related to it than the closest words. However, this information can also be learned by the network itself by increasing the weights of the closer context words with respect to the weights of those further away. It was therefore not implemented as it adds additional computation cost.
- Adding phrases to the vocabulary: The words "Air" and "Canada", for example, have a different meaning on their own than the combination of them "Air Canada" would have. Adding phrases to the vocabulary that are composed of several words increases the quality of the word vectors significantly. This is, however, of no concern in the type of text present in this project's corpus and was thus not implemented.
- Negative Sampling: Since the input and output to the network are One-Hot-Vectors, the number of weights in the network is proportional to the length of the vocabulary size V . For large vocabularies this means that the network takes a long time to be trained while most of the training effort goes into tweaking the $V - 1$ entries with "0" in the One-Hot-Vector. This is not necessary and the solution is to only update the weights to a few of the $V - 1$ negative words plus, of course, the weights to the one positive entry in the vector. The authors of Word2Vec state that updating 5 – 20 negative words gives good results in small data sets while 2 – 5 is sufficient in large corpora. In this project, a medium number of 5 negative words was chosen.

The resulting vectors of Word2Vec achieved groundbreaking scores in the field of NLP. Thanks to its simple architecture, the authors could train word embeddings on text corpora with vocabulary sizes that were several orders of magnitude larger than up to that date. The obtained embeddings then set a new standard in both semantic and syntactic word relationship and analogy tasks with the outstanding result of achieving a 50% increase in the Spearman's rank correlation compared to previous results. Moreover, the vectors were found to have successfully encoded linguistic patterns and symmetries in a way that mathematical operations on the vectors yielded meaningful results both in the machine's language vector space as well as in human language. For example, subtracting $vector("Man")$ from $vector("King")$ and adding $vector("Woman")$ to it does indeed result in a vector that is closest to $vector("Queen")$ in cosine distance [52] [55].

3.3.2 GloVe

While it was a somewhat surprising byproduct of Word2Vec that the meaning in the word embeddings is encoded in distances and offsets in vector space, GloVe intentionally seeks to implement this feature [54]. It does so by combining both the global and local context approach - it counts in the global frame, the document, how often local word co-occurrences appear. However, the authors argue that it is the ratio of co-occurrence probabilities rather than the probabilities themselves that bear the valuable information. The ratio of counts is thus normalized, smoothed with a logarithm and stored in a VxV co-occurrence matrix which is then given as input to GloVe. There, the matrix is factorized to obtain a lower dimensional matrix where each row represents the embedding of one word.

The results are very similar compared to the results obtained with Word2Vec, for a graphical illustration of the word analogy in GloVe see Figure 14. While the authors of GloVe claim to score higher on various performance measures [54], other research evidence points in the opposite direction [56]. However, the differences in performance of both approaches are not substantially high and depend a lot on the specific task which they are compared on. The competition is still ongoing, however, it has been shown both theoretically as well as empirically that while both approaches have different starting points they both arrive at the exact same solution. That is to say, while GloVe explicitly factorizes the word co-occurrence matrix, the Word2Vec model does so implicitly as well [57] [58]. The choice of word embedding for this project thus fell on the Word2Vec model for its easy implementation, fast training and degree of recognition. An approach with another word embedding model, however, can be an interesting extension for future work on this project.

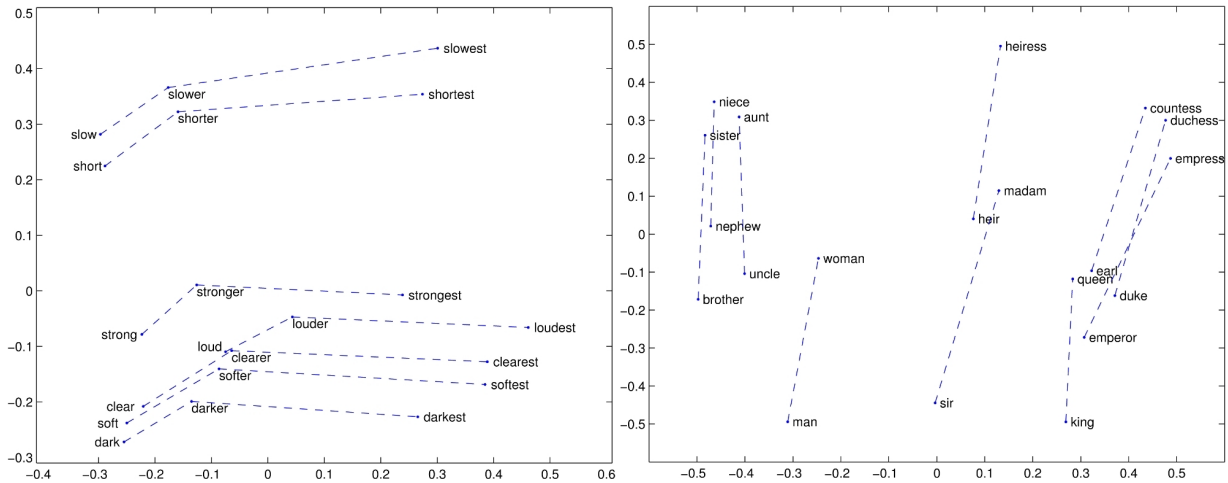


Figure 14: Word vectors obtained with GloVe. The model successfully captures the relatedness of words both in syntactic (left) as well as semantic (right) tasks: Vector differences between two related pairs of words are parallel. Source: [54]

3.3.3 N-grams with Naive Bayes

The collaborating team of Johan Frid et al. used another word embedding method which they describe as "bag-of-n-grams with a Naive Bayes component" [6]. The concept follows the NBSVM approach by Wang et al. [59] and Mesnil et al. [60]. The idea behind this is the following: In the first step, all possible n-grams are extracted from the sentences, from $n = 1$ to $n = 4$. Here, n-grams are sets of n adjacent words in a sentence. They then compute a log-ratio vector between n-grams that were extracted from sentences classified as positive and those of the negative sentences. Note that they train a Support Vector Machine (SVM) for only one category at a time which makes it possible to unambiguously classify sentences as positive or negative. These vectors serve then as the input for the SVM.

This approach thus uses information about the task of the classifier, the classification labels, already for the representation of the input to the classifier. This makes it very effective for the problem at hand while leaving little room for generalization.

3.4 Support Vector Machines

Support Vector Machines (SVMs) are an alternative machine learning algorithm to ANNs. They are used by our collaborating team, Johan Frid et al. [6] and are discussed here in order to make reasonable comparisons.

In their present form, they were proposed by Cortes and Vapnik [44] in 1995 while the first pioneer work was already done by Vapnik as early as 1963. Just like ANNs they are applied on two major types of problems, classification and regression. Their approach to solve these problems, however, is fundamentally different and will be shown on the example of a classification problem [45]:

Figure 15 shows data points of two linearly separable categories indicated as blue stars and red dots. As discussed in Chapter 3.1, an ANN finds a hyper-plane to separate the two categories by **minimizing the training error**, i.e. by minimizing the number of misclassified data points. The result is one out of many possible separation lines, indicated as the blue lines, depending on stochastic factors like the initialization of the weights, the order in which data points are presented, etc.

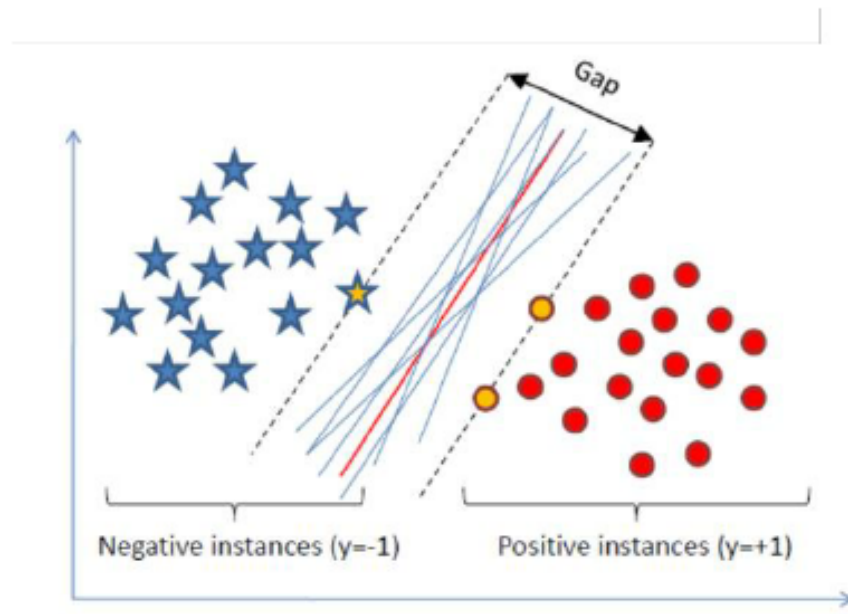


Figure 15: A linearly separable classification problem. The grey lines are examples of solutions that may be found by an ANN since they all minimize the number of misclassifications. The optimal solution, the red line, is found by a SVM by maximizing the margin, the distance between the separation line and the closest data points (yellow). Source: [46]

The result obtained by a SVM, on the contrary, is not just any possible solution but the optimal solution, indicated as the red line. It is obtained by **maximizing the margin**, where the margin is the distance between the separation line and the data points closest to it. These data points (colored in yellow) are called the support vectors and they are the only fraction of the data that is actually used to describe the separation line. Note that SVMs are linear classifiers and can only categorize data that is linearly separable. However, state-of-the-art SVMs use a method [44] to classify non-linear problems referred to as the "*Kernel Trick*". The idea behind this method is to map the data onto a higher dimensional space where it is then separable by a linear SVM in the higher dimensions, see Figure 16. The choice of the Kernel function depends on the problem at hand with the Gaussian Kernel being the most commonly used. In NLP polynomial Kernels are often used as well, since they effectively model occurrences of pairs (triples, quadruples, ...) of words [48].

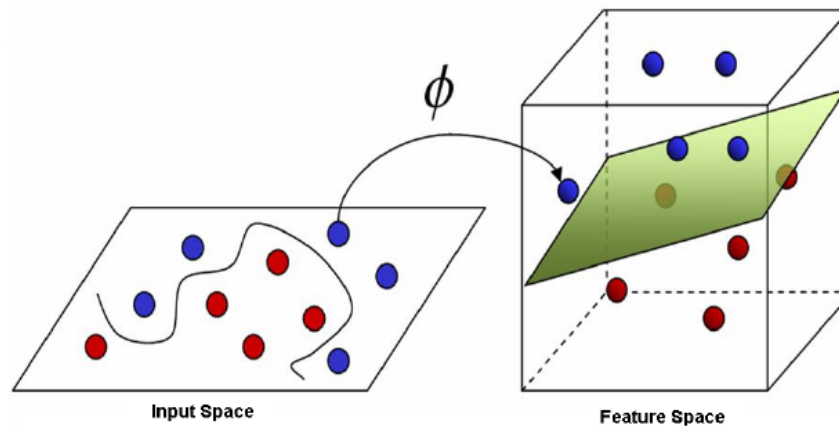


Figure 16: Visualization of the Kernel Trick: The non-linear separable data is mapped onto a higher dimensional space by the Kernel function Φ . In this space it is now linearly separable. Source: [47]

A comparison between the ANN and SVM approach when applied to NLP yields the following:

- Up until recently (\sim 2012/2013), SVMs usually outperformed ANNs on prediction accuracy as shown in [49] and [50]. The beginning of the deep learning era and the application of CNNs in NLP, however, are starting to shift the predominance in this competition [5].
- SVMs are only applicable to binary classification tasks whereas ANNs can deal with any number of categories. For multi-class classification, several SVMs need to be used in sequence.
- SVMs are much slower than ANNs in problems that have a lot of input data. The use of billions of words as input, as it is the case in some NLP tasks, can thus become problematic.
- Once trained, SVMs are faster at predicting than ANNs. This is because an input has to be passed through the whole network in the case of ANNs whereas a SVM only needs to decide which side of the decision hyper-plane the data point lays on.
- SVMs take all the data as input at once whereas ANNs learn online. If new data is available an ANN can thus be updated while a SVM has to be trained from scratch.

In this project, mainly the first and second point are going to be examined. While the SVM is expected to perform better on prediction accuracy, it is unclear how severe the difference for this particular problem and data set is. If the loss in accuracy is acceptable, the use of ANNs might be favorable in the light of the other advantages they have. Secondly and even more importantly, the second point poses a substantial restriction on SVMs. While it is still possible to do multi-class categorization for a small number of classes by

simply combining many binary SVMs together, this becomes less and less feasible for large numbers of classes. For this reason it is of great interest to develop ANNs that can compete with the performance of SVMs since ANNs have no upper limit for the number of classes they deal with.

4 Implementation details

4.1 Software

The programming language that was used for this project is PYTHON (version 3.6.2). The designing, training and analyzing of the various Artificial Neural Networks was done with the library KERAS (version 2.1.2) [66]. KERAS is an open source library that was developed for user-friendly and fast experimentation with deep ANNs. It is capable of running on top of several machine learning libraries such as TENSORFLOW, MICROSOFT COGNITIVE TOOLKIT, THEANO and MXNET. The choice in this project fell on the TENSORFLOW BACKEND (version 1.3.0) [67]. For more details on the software, see Appendix A.4.

4.2 Reproducibility

Over the course of this project reproducibility turned out to be, somewhat surprisingly, a major problematic issue. The results and performance of networks with the same parameters suffered from significant fluctuations. The reasons for this proved to be rooted both in the particular problem and data set at hand as well as generally in the software that was used:

- Problem specific
 - The data set in this project is relatively small (1006 sentences) compared to the number of categories (14). Additionally, the distribution of the sentences over the categories is considerably uneven. This results in some categories only being represented by a very small number of sentences with the least frequent category, Assessment of end-points, appearing only 16 times, see Chapter 2.2 Table 2. The performance of the network thus depends heavily on the training/test set split and also on the order within the respective split in which the data is presented to the network.
 - The error landscape proved to exhibit a structure with many local minima that the network could get stuck in. For this matter, see the flat passages in the error plots in Chapter 5. The final state of the network can thus be strongly influenced by its starting point, i.e. the random initialization of the weights.
 - The randomness in the Dropout method, see Chapter 3.2.1, also influences the functioning of the network in an unintended way: If during the training phase one of the rare categories is presented to the network and the nodes that are

necessary to detect this category happen to be dropped out, the network does not get many more chances to train on this category. The odds for this to happen are thoroughly considerable due to the relative small sizes of the networks.

- Software specific
 - The software used in this project is KERAS with TENSORFLOW backend. As described in this post on github [63], the TENSORFLOW backend for KERAS does normally not allow for exact reproducibility of results for certain hash-based operations. Other backends, however, as for example THEANO backend do not have this limitation. The reason for this is that TENSORFLOW uses multiple threads or cores. This becomes an issue when float values are rounded and shared over the various threads and leads to irreproducible outcome. While in most cases the effect is marginal, the networks in this project which are quite sensitive due to the reasons described above are substantially influenced by it.

The solution to discarding these two sources of fluctuation is the following:

During the optimization of hyper-parameters, the data split, the order of sentences within the splits and the initialization of weights were seeded. This is to make sure that the choice of hyper-parameters is based solely on the signal and not on fluctuations. For the estimation of generalization performance, however, this seed was removed to obtain a result that is valid not only for one specific set of initial weights and data split.

The fluctuations that are due to software specific reasons were removed for the entire duration of the experiments, i.e. hyper-parameter search and estimation of generalization performance. This one done as described in the KERAS FAQ [64]: The Python hash is seeded and then TENSORFLOW is forced to use only one parallel thread. The drawback is the significantly reduced speed due to the usage of only one parallel thread.

To compare the effect of the different types of fluctuations, see Table 3.

Table 3: The effect of the different types of fluctuations on the combined categories CNN (Chapter 5.1.2) with respect to the validation AUC and loss. Mean and standard deviation are calculated from 10 test runs. The AUC value is the average value of all 14 categories. Filtering out the software specific fluctuations reduces the overall fluctuations of the loss by 19% and AUC by 12%.

	AUC		Loss	
	Mean	Std	Mean	Std
All fluctuations present	0.82	0.017	6.0	0.16
No software specific fluctuations	0.82	0.015	6.0	0.13

4.3 Training parameters

In Table 4 all parameters of the different parts of the networks are listed. The parameters which are specified are kept constant with the indicated value throughout the entire project if not explicitly stated differently. An unspecified parameter is subject to the researcher's modifications during the optimization process. The reasoning behind the usage of different optimizers for CNNs and LSTM units is discussed in the appendix A.1.

Table 4: Parameters in the experiments. Where ever specified, the particular value was used during all the experiments. The parameters that are left blank were subject to the optimization of the various models.

General		Word2Vec	
# K folds	5	embedding dimension	60
batch size	20	context window	10
# epochs		min word frequency	1
learning rate		down-sampling	10^{-3}
		static word vectors	False
CNN		LSTM	
filter sizes	(1, 3)	# LSTM, layer 1	
# filters per size		# LSTM, layer 2	
# hidden nodes		# hidden nodes	
L2-regularization		L2-regularization	
input dropout		clipnorm	1.0
optimizer	Adam	optimizer	AdaGrad
MaxPooling size	2		

5 Results and discussion

The fundamental results of this project are summarized in Table 5. The CNNs for individual categories perform very similar to the SVMs from the collaborating research team [6]. The combined categories approach naturally performs weaker than the individual categories. It is however interesting that the differences in performance between both approaches vary considerably between the categories. The attempt of finding a pattern in this is rather speculative. It is apparent though that the very frequent categories like "Exposures" and "Outcomes" suffer a greater loss in performance in the combined network than the less frequent ones. This is most likely due to the relatively high benchmark that the individual network can set for these categories because of their high frequency. Generally one needs to note, however, that while the performance does loosely scale with the frequency, this correlation is not as strong as one might expect. The category "Subgroups", for example, has a medium frequency and scores significantly weaker than all the low-frequency categories.

Table 5: AUC scores of CNN and LSTM networks, each for individual and combined categories. The values for each category are obtained by averaging over 30 runs with 5 folds each, the brackets indicate the Standard Deviation (StD) of this average. The error of the all-category mean is propagated from the single StDs. The AUC scores of the SVM approach are taken from Johan Frid et al.[6] for comparison.

Category	Frequency	CNN		LSTM	SVM
		Indiv. cat.	Comb. cat.	Indiv. cat.	Indiv. cat.
Aim	91	0.94 (0.02)	0.91 (0.02)	-	0.97
Exposures	788	0.93 (0.01)	0.80 (0.01)	-	0.91
Outcomes	686	0.91 (0.01)	0.72 (0.01)	-	0.93
Study design	31	0.85 (0.01)	0.81 (0.02)	-	0.79
Inclusion criteria	31	0.84 (0.03)	0.81 (0.02)	0.75 (0.13)	0.87
Actual size	154	0.91 (0.01)	0.88 (0.01)	-	0.90
Several cohorts	62	0.89 (0.01)	0.78 (0.02)	-	0.84
Subgroups	142	0.80 (0.01)	0.64 (0.02)	-	0.80
Follow-up period	68	0.90 (0.02)	0.86 (0.01)	-	0.91
Ass. of end-points	16	0.85 (0.05)	0.81 (0.03)	-	0.81
Statistical methods	81	0.90 (0.04)	0.81 (0.02)	-	0.96
Results	323	0.95 (0.02)	0.90 (0.01)	-	0.95
Conclusions	164	0.87 (0.03)	0.84 (0.01)	-	0.90
Methods	313	0.95 (0.01)	0.92 (0.01)	-	0.98
Mean		0.89 (0.09)	0.82 (0.06)	-	0.89

It is obvious that the categories with lower occurrence have a greater fluctuation in performance as indicated with the StD in brackets. An interesting observation to point out is that "Study design" is the category with the highest improvement compared to the

SVM approach. It is the example category with which the network architecture for all categories was determined. The networks for the other categories were optimized individually only on their respective regularization, see Chapter 5.1.1. This suggests that for further performance improvements in the future, individual network architectures for each category should be considered.

The low score of the RNN (LSTM units) shows the difficulties that are associated with training this type of network. Its unstable nature and high dependence on exact hyper-parameter optimization is even more apparent when looking at the high StD of the AUC score. Since this RNN could not be optimized thoroughly by the end of this project, its AUC score should not be interpreted as a final result but rather as an indicator of its instability compared to the comparably robust CNN.

5.1 Convolutional networks

Solving the categorization problem of the sentences with networks that have a convolutional structure was the initial and main task of this project. It was done in two approaches: Setting up individual networks for each category and finding one combined network for all categories together. The procedure to find the optimal network hyper-parameters was the same for both approaches.

First, the parameters concerning the physical network size were determined. These include the size of the convolutional filters, the number of filters per size and the number of hidden nodes in the dense layer after the convolution layer. Note that the parameter of filter sizes was added to the set of optimizable hyper-parameters only later on during the project. The initial assumption of which filter sizes would be beneficial, however, proved to be right (filters of size "1" and "3"), see Table 6 in Chapter 5.1.1. The other two parameters of the network architecture, the number of filters and the number of hidden nodes, were optimized in a random search. For this purpose, networks with random combinations of the two parameters, each drawn from a certain range, were tested on their training AUC and training loss. The combination with the smallest architecture that still managed to optimize the training AUC to one and the training loss to zero, was picked as the lower limit of necessary complexity. The actual size of the network architecture was then chosen by adding some extra complexity (see the respective chapters for a quantitative discussion) to this lower limit in order for the network to have enough degrees of freedom to perform well on the validation data as well.

The second step after determining the network size was then to apply regularization to prevent over-fitting, see Chapter 3.2. Because of the relatively small network sizes, L2-regularization was preferred over Dropout in order to not drop out the essential nodes in an already small and restricted network, see Chapter 3.2.2. Since the optimization process for network regularization included thus only one parameter for most networks, the L2-regularization, the search was done in a shrinking grid instead of random combinations. Note that the network architecture was optimized for the network to be equipped with the necessary degrees of freedom to categorize sentences by capturing features within them.

This was therefore done on the training set to provide evidence that the network is capable of the task at all, as described above. The purpose of the regularization, however, is to improve the network's generalization performance. It was therefore optimized with respect to the validation AUC. Doubts about optimizing the regularization on the validation data are addressed in Chapter 2.4.2: The whole optimization process was done with seeded data splits and weight initialization because of high fluctuations. The training on validation data was thus only done for a specific data split. The generalization performance, however, was obtained by running 30 unseeded simulations which were then averaged.

5.1.1 Individual categorization

In this part, individual networks are optimized for each category in order to obtain results that are directly comparable to the SVM approach [6]. The final results are listed in Table 5. The experiments were done with the parameters indicated in Table 4 where $\#epochs = 500$ and $learning\ rate = 0.0001$. The discussion focuses on the example of the category "Study design" which is the most difficult category according to the results of the SVM. All other categories have similar and better results and can be seen in the appendix A.2.

Initially, the parameter of filter sizes was not subject to optimization. Instead, filters of size one and three were picked with the following reasoning: The filters of size one treat every word on its own, allowing the network to extract the necessary first order information on a word to word basis. While this is intuitively understandable the choice of the second filter size, three, is more complicated. As described in the end of Chapter 3.1.1, certain words within a sentence are related to each other. The example sentence that was given there would, however, rather suggest a relation between pairs of words, i.e. a filter of size two. The reasons for choosing the size three instead of two are the following: The example sentence is very simple while the sentences in this project are of higher complexity and length. Two adjectives in front of one noun or a noun that has both an adjective and a related verb are the expected complexity that the network has to extract information of. Additionally, stop words and abbreviations are not removed which means that the next meaningful word is often not right in the immediate neighbourhood.

The optimization of the network parameters was thus done with the fixed filters of sizes one and three. Later on, this assumption was tested on the optimized combined network in Chapter 5.1.2. The results of this test, however, will be given at this point already for a more clear structure. The assumptions about the optimal and necessary filter sizes being (1, 3) were proven right which can be seen in Table 6. The filters of size two did not perform as well because they did not have the necessary reach to capture all word relations. On the other hand, the filters of size four seemed to be too large and diluted the content. A comparison between sizes (1, 3) and sizes (1, 2, 3) shows that the size two does not result in an increase of performance.

Table 6: AUC performance of the network for different filter sizes. The combination of filters of size one and three works best to capture the information in the sentences. Network parameters: Combined network from Chapter 5.1.2.

	(1, 2)	(1, 3)	(1, 2, 3)	(1, 4)
AUC	0.77	0.83	0.82	0.78

The necessary network architecture was determined in a random search without regularization as described in Chapter 5.1. The results can be seen in Figure 17. In order to have a classification performance on the training data of $AUC \approx 1$ and for the training loss to be reduced to $loss \approx 0$, the network needs a complexity of at least five hidden nodes and one filter per filter size. While the size of the hidden layer is very problem specific as it is responsible for the categorization, the convolutional layer which acts as a feature extractor should be comparable to other sentence classification tasks. It is therefore surprising that only one filter per size is needed in this task while Yoon Kim [5] uses a network with 100. This is a first indicator for the condition of the word2vec word vectors which will be further discussed in Chapter 5.3. Yoon Kim used vectors that were trained on a 100 billion words corpus while in this project only a corpus of 29306 words was accessible. The resulting vectors are thus not fine tuned enough to allow for many filters to each extract different meaningful features.

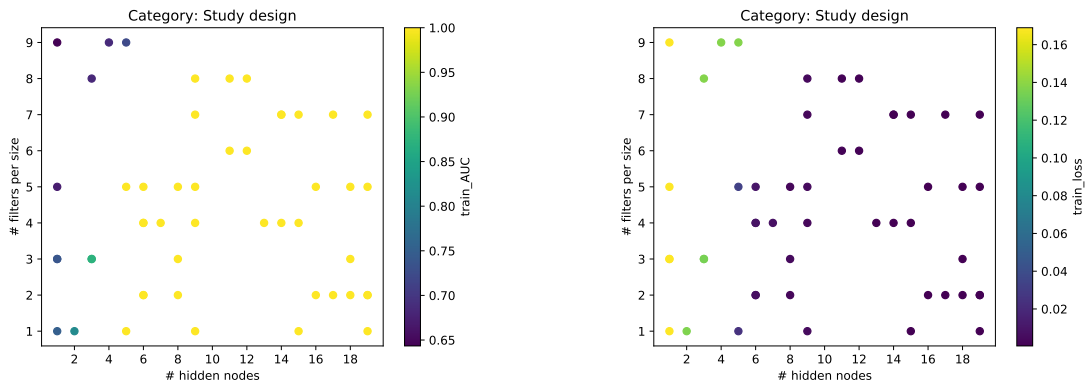


Figure 17: Random search to determine the necessary complexity of the network in the convolutional and hidden layer. Both for maximizing the training AUC (left) as well as for minimizing the training loss (right) the network needs at least five hidden nodes and one filter per size.

The de facto network size was then picked to be above that minimum size to ensure that the network had enough complexity to perform well not only on the training data but also on the validation data. The number of filters per size was thus increased from one to two and the number of hidden nodes was raised from five to eight. This was feasible because the computational cost for such small networks, even if increased in size,

are modest. The final network architecture was thus: An input matrix of 121 (number of words + padding times) times 60 (dimension of word embedding) followed by a layer of convolutional filters (two filters per size one and three), each followed by max pooling of stride two whose outputs are then concatenated and fed into a dense layer of eight hidden nodes. The output layer consists of only one node since the categorization is binary. Note that this structure was obtained by optimizing on the example category of "Study design" which is the most difficult category according to the SVM approach, see Table 5. The categorization of the other, arguably more easy classes technically required even smaller networks, see the appendix A.2.1. However, for better consistency and comparability, this network architecture was used for all categories throughout the project.

Having the same network architecture for the different categories, however, creates the need to individually regularize them to take each category's different amount of excessive degrees of freedom into account. Regularization was done with L2-regularization in most cases while Dropout was avoided as described in Chapter 3.2.2. An exception was the category "Actual size" which is discussed further below. The results can be seen in Figure 18. The performance of the network improves steadily with increasing L2-regularization because over-fitting is successfully suppressed. At a certain level, the regularization dampens too strongly and results in decreasing performance. The second part of Figure 18 shows that Dropout has a negative effect on the overall performance and should be avoided. This is the case for all categories except for "Actual size". The plots for the L2-regularization searches for all other categories can be found in the appendix A.2.2.

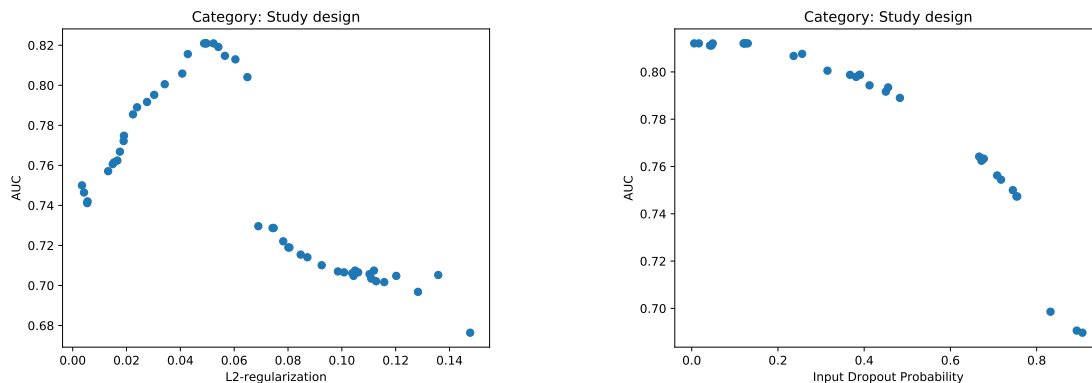


Figure 18: Performance improvement of the network with L2-regularization (left). The addition of Dropout, on the other hand, does not have a positive effect (right).

In the case of category "Actual size", the optimal performance is achieved for zero L2-regularization, see Figure 19. Instead, a Dropout of about 90% applied to the input word vectors yields a significant improvement. There are two observations to be made here: First, the phenomenon of high performance for very low or zero L2-regularization is present in many categories, see the appendix A.2.2. For all those categories, however, there is a

second performance maximum for non-zero L2-regularization. The existence of two maxima is somewhat surprising as one would expect the performance to increase up to a point where all overfitting is avoided followed by a steady decrease due to over-regularization. It is not entirely clear what this second maximum is due to. One possible explanation is that there are numerical reasons for it as the error landscape is highly affected by the existence or non-existence of regularization. Another possible reason is the usage of the same L2-regularization on all layers of the network (the embedding, convolutional and dense layer). Different parts of the network can be optimized by different amounts of regularization and the usage of one global parameter can result in several maxima in the overall performance. The fact that there is no second, non-zero maximum for category "Actual size" within the search range means that the network was left unregularized in this case.

This leads to the second observation. The network for "Actual size" which was unregularized with respect to L2-regularization can be improved with Dropout on the input instead. The fact that it reaches its optimal performance when 90% of the input is dropped adds to the evidence that the input word2vec word embedding did not manage to encode the desired information to its full extent, see Chapter 5.3.

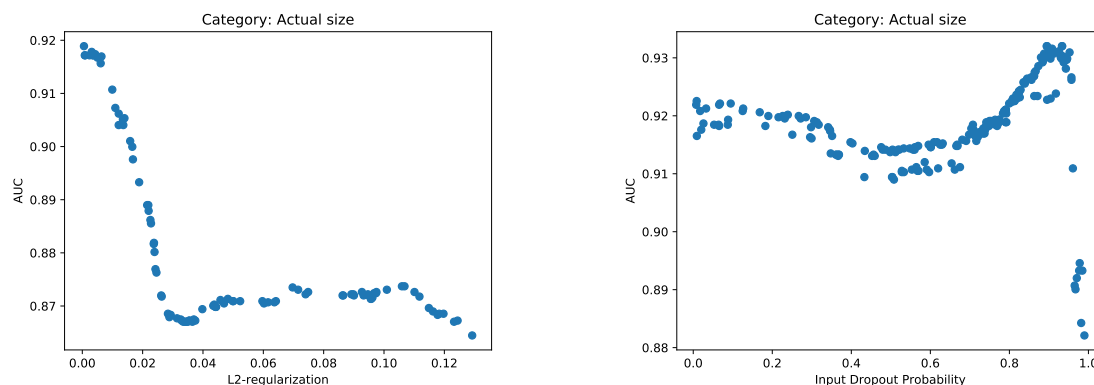


Figure 19: In the case of category "Actual size", the performance does not improve with the addition of L2-regularization (left). A very high Dropout rate on the input, however, results in the increase of AUC (right).

Once the networks were optimized in both size and regularization, 30 iterations with each 5 folds were run for each category in order to estimate the generalization performance. The final results given in Table 5 are obtained by averaging over these 150 runs each. Note that the 5 fold runs in one iteration can each yield quite different results in the case of rare categories, see Chapter 2.4.2. The example category "Study design" is such a rare category and its different results in loss and AUC for best and worst 5-fold run within one iterations can be seen in Figure 20 on the left and right respectively. The more frequent categories are more consistent within the 5-fold runs as can be seen on some examples in the appendix A.2.3.

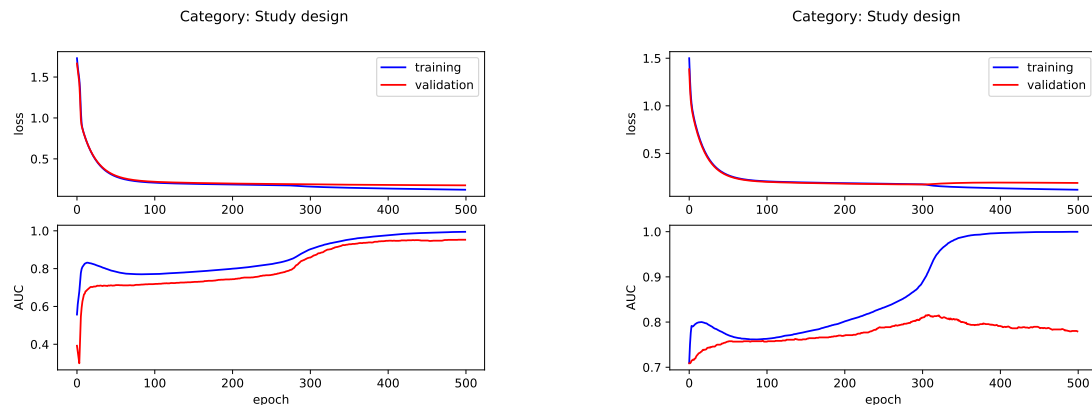


Figure 20: K-fold runs with best (left) and worst (right) results within one iteration for the rare category "Study design". While the loss is minimized successfully in both cases, the AUC performance depends on having enough positive sentences in the data split and thus varies significantly.

In the case of "Actual size", Dropout had to be applied instead of L2-regularization. The results of this and the reason why Dropout was avoided whenever possible in the other categories can be seen in Figure 21. Both in the best (left) and worst (right) 5-fold run, the minimization of training and validation loss is a lot less smooth than for the L2-regulated networks. Also, it takes many more epochs (300-400 compared to 80) until the training loss is minimized to close to zero. Instead, the loss is stuck in a plateau for about 150 epochs until the network finds a way to decrease it further. The use of input Dropout, especially to this high extend (90%), should thus be avoided whenever possible. However, the AUC performance is stable and reaches results comparable to the SVM after averaging over the 150 runs, see Table 5.

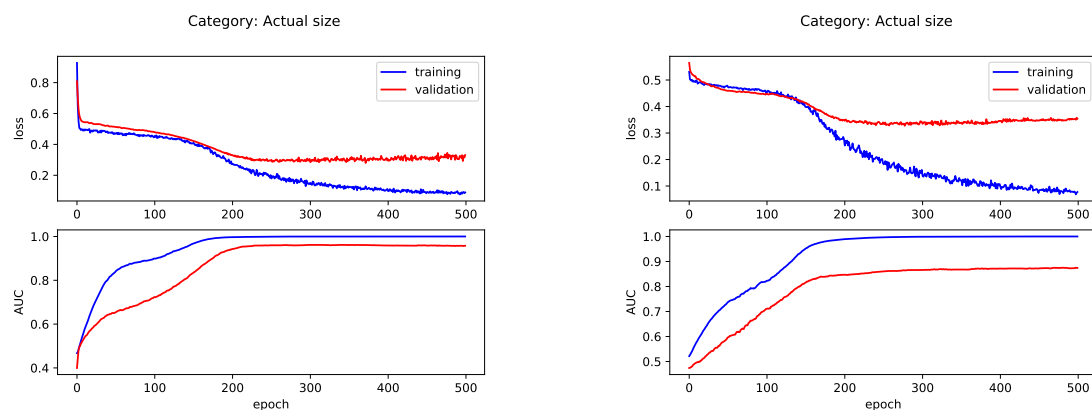


Figure 21: 5-fold runs with best (left) and worst (right) results within one iteration for the category "Actual size" which was regulated using 90% input Dropout. The minimization of loss is less smooth and takes longer than for the L2-regulated networks.

5.1.2 Combined categorization

After training individual networks for each category in order to directly compare the results with the SVM approach, a single CNN for all categories together was set up. A combined network is favorable over individual networks for future applications. However, the question to be investigated here is how much loss of performance such a more complex combined approach would suffer compared to the individual classification. In this chapter, the AUC score for the combined network is understood to be the average AUC of all categories. The experiments were done with the parameters indicated in Table 4 where $\#epochs = 200$ and $learning\ rate = 0.0001$.

The optimization of hyper-parameters was done analogically to Chapter 5.1.1: The necessary network complexity was determined by minimizing train loss and maximizing train AUC. The results can be seen in Figure 22:¹ While the train AUC could be maximized to one, the train loss could not be pushed under a certain level (≈ 3.5) even with very large architectures. This suggests that the problem is not thoroughly solvable by adding complexity in width. One solution might be to expand the network further into its depth by adding supplementary layers as it is done in Deep Learning. In this project, however, the option of Deep Learning was dropped in favor for a comparative investigation into Recurrent Neural Networks.

The minimum complexity of the network in order to achieve a train AUC close to one as well as a minimum train loss is then 30-40 hidden nodes and 3-4 filters per size. Analogically to the individual networks, extra complexity was added and the final architecture of the combined network consists of 50 hidden nodes and 4 filters per filter size. Also, the network was optimized on validation AUC using L2-regularization, see appendix A.3.1.

¹Note that here they are displayed in separate plots for filters and hidden nodes instead of a 3D color plot for better visibility reasons.

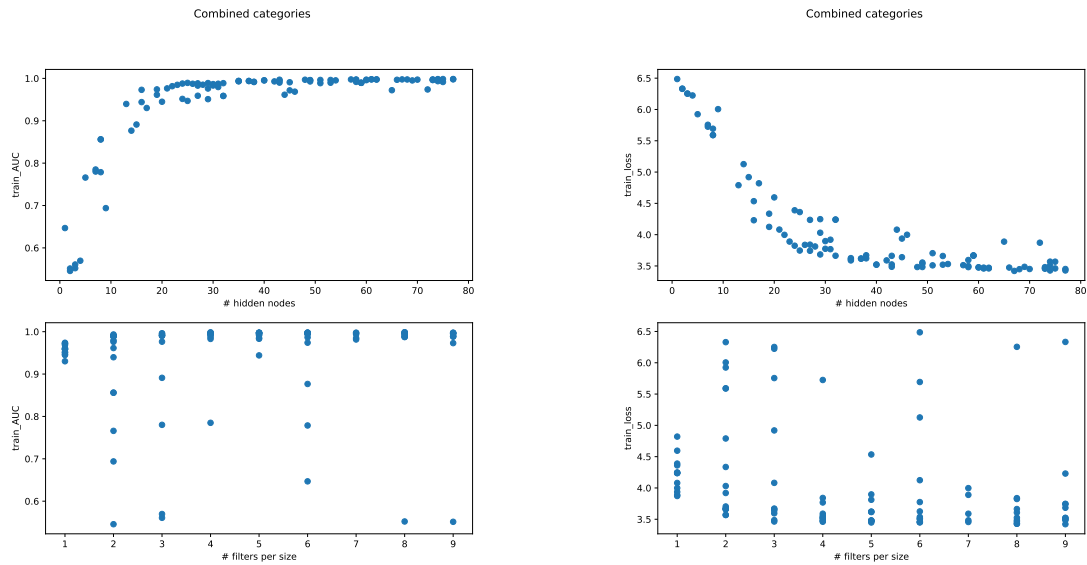


Figure 22: Architecture search for the combined network. An AUC of one for the training data is reached at minimum 30-40 hidden nodes and 3-4 filters. The training loss converges around 3.5 and can not be pushed entirely towards 0. Adding additional complexity to the minimum leads to the final decision of architecture: 50 hidden nodes and 4 filters per size.

The so optimized network was then trained 30 times with 5 folds each. The results of one such exemplary run can be seen in Figure 23. While the loss does decrease steadily it can not be pushed as low as in the individual classification case. However, the effect of over-training could be balanced out successfully as the validation loss does not increase again after reaching its minimum. The final AUC scores are stable after about 75 epochs which corresponds to the last significant drop in loss. The final averaged results of the 150 runs can be seen in Table 5.

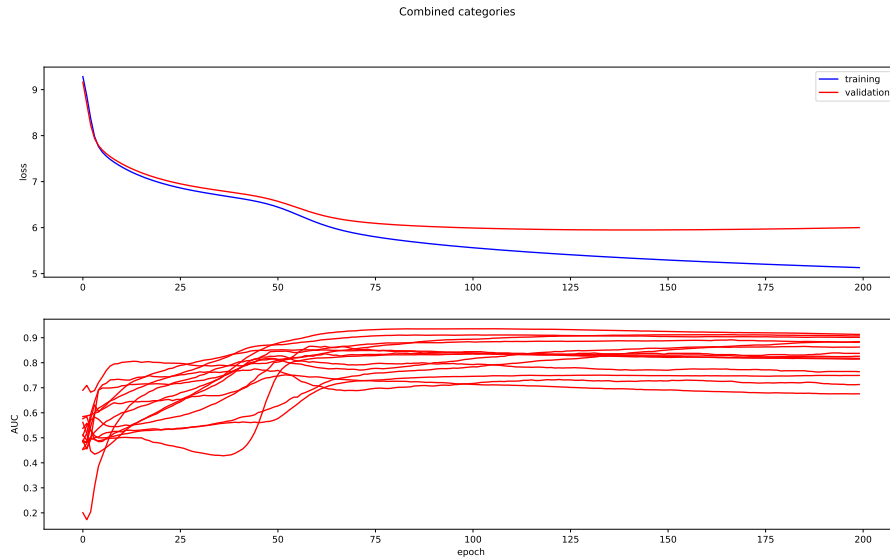


Figure 23: Exemplary run of the optimized network for combined categories. Training AUC scores are omitted for better visibility. For an allocation of each validation AUC plot to its respective category, compare Table 5.

5.2 Recurrent networks

The experimentation with RNNs, more specifically LSTM units, was added as an extension to this project. RNNs are known to be very sensitive to the choice of hyper-parameters [21] and are therefore quite unstable compared to CNNs. This behaviour turned out to be even more emphasized with the somewhat challenging data set in this project (high differences in category frequency, poor Word2Vec representations, etc.). The experimentation was thus constricted to the individual classification task which was done exemplarily on the low frequency (31/1006) category "Inclusion criteria".

As described before, the network size was optimized on the training AUC and training loss on a seeded data split. The necessary regularization of this network was then found by optimizing with respect to the validation AUC. The results can be seen in Figure 24. In order to reach a training loss of zero, only one LSTM node was needed, followed by 40 hidden nodes. The validation performance has several maxima for different amounts of L2-regularization, our choice fell on the central maximum at 0.00012.

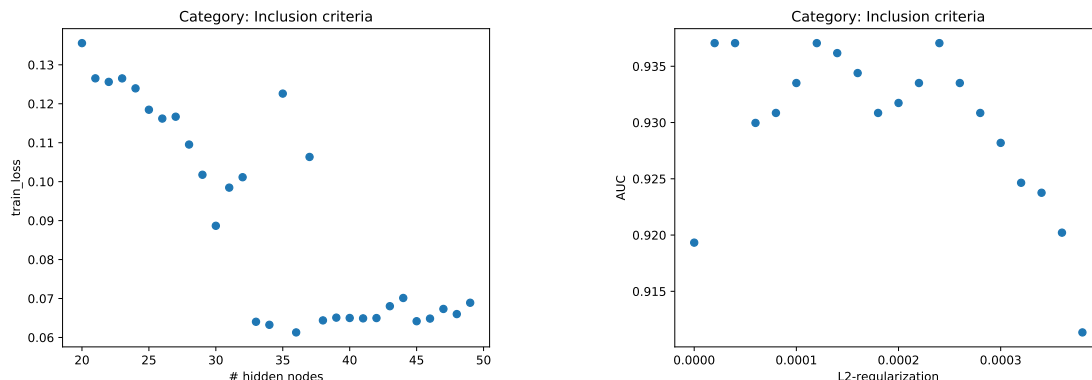


Figure 24: Architecture and regularization optimization for category "Inclusion criteria". With only one LSTM unit, the necessary hidden dimension is around 40. As for the L2-regularization, 0.00012 was picked out of the three possible maxima.

The unstable nature of the RNN first shows then when removing the seed of the data split in order to obtain the generalization performance. Figure 25 shows the best and worst run out of one 5-fold iteration. Even on the best run, the network which had in fact been optimized before shows clear signs of over-training such as increasing validation loss and decreasing validation AUC. The worst run shows the same signs of over-training. In addition, however, the loss and consequently also the AUC curves are not smooth but show sudden peaks and edges. This is most likely due to exploding gradients, a common problem in RNNs. Note that the effect in this figure has already been partly attenuated by applying a so called "clipnorm" to the gradients in the network which cuts off excessive gradients. The effect without the clipnorm renders the results meaningless.

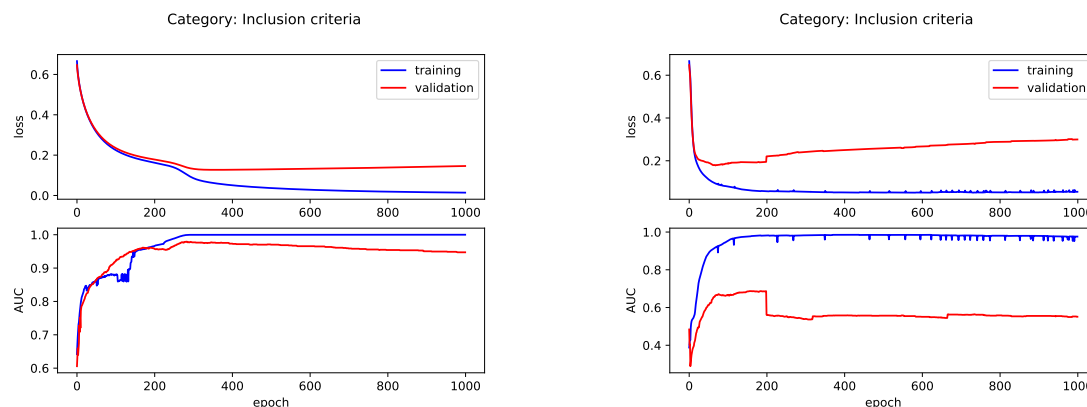


Figure 25: Architecture and regularization optimization for category "Inclusion criteria". With only one LSTM unit, the necessary hidden dimension is around 40. As for the L2-regularization, 0.00012 was picked out of the three possible maxima.

As mentioned in [21], the choice of hyper-parameters is crucial to training well working

RNNs. According to the authors, especially the batch size and number of hidden nodes is important. The Keras manual points out that the choice of optimizer is also decisive when using RNNs. While a comparison of optimizers was done and can be seen in the appendix A.1, there was no time in this project to further look into the exact fine tuning of the LSTM hyper-parameters or other categories. The LSTM result in Table 5 should thus not be seen as a final measure of the capabilities of RNNs on this NLP task. It rather illustrates the difficulties and challenges of RNNs in this field.

5.3 Word2Vec performance

The performance and quality of the Word2Vec word representation was an important issue throughout the whole project. A good word embedding is essential for the network to be provided with meaningful input data. The Word2Vec method is one of the most sophisticated and promising word embedding of state-of-the-art NLP. However, for its whole potential to be exploited it needs to be trained on very large corpora of text and a variety of pre-processing methods should be applied, see the discussion further below.

There are indicators that the quality of the Word2Vec word representations trained in this project was unsatisfactory:

- In the individual categorization with CNNs, Chapter 5.1.1, only one filter per size is needed whereas 100 were used by Yoon Kim in his network [5]. This is an indicator that the word vectors in this project did not contain enough fine tuned information for so many filters to each extract meaningful features.
- In the case of category "Actual size", the CNN was regularized with Dropout on the input instead of L2-regularization. The optimal rate for this dropout was shown to be as high as 90%. This means that most parts of the word vectors contained dispensable information.
- The optimal dimensionality of the word embedding was empirically shown to be 300 in the original Word2Vec paper [52]. It was reduced to 60 in this project without significant change of performance.
- The Word2Vec word vectors were trained before the experiments with the actual networks. However, during the training of the networks they were still subject to further training as the authors in [5] pointed out that these continuous adjustments can lead to improvement of performance. During the experimentation phase with the LSTM units, however, a test with static word vectors was done as well. While the network which was trained with variable word vectors needed only one LSTM unit and 40 hidden nodes, the network with static word vectors had to be as large as two layers of each 100 LSTM nodes followed by 50 hidden nodes. This suggests that most of the training and thus the main part of the classification task in this project consists of finding good vector representations for the words.

All these observations suggest that the Word2Vec embedding did not represent the words in an optimal way. The reasons for this are the following:

First and most importantly, the word vectors in [52] and [5] were trained on text corpora of billions of words while the text corpus in this project was only of size 29306 words. Many of the words thus appeared only very few times in the corpus and training a meaningful representation for them based on this little context is challenging. Additionally, most of the pre-processing methods for Word2Vec were not applied for various reasons connected to the specific data at hand as discussed in Chapter 3.3.1. These methods, especially the deleting of rare words and the negative sampling technique, can improve the quality of the vectors significantly as stated by the authors [55]. Possible future work connected to improving the Word2Vec word embedding for the data at hand is discussed in Chapter 7.

6 Conclusion

In this Natural Language Processing (NLP) project sentences from abstracts of medical articles were classified using Convolutional Neural Networks (CNNs). The usage of Recurrent Neural Networks (RNNs) to tackle the same problem was added as an extension. In both network architectures, the inputs consisting of words were embedded using the Word2Vec representation. The results were then compared to the work of our collaborating team which used Support Vector Machines (SVMs) for the same classification problem.

In the CNN approach, the classification was done in two different ways: With individual networks for each category and with a combined network for all categories together. The individual networks score an average AUC of 0.89 ± 0.09 which is exactly what was achieved with the other team's SVMs. It was thus shown that Artificial Neural Networks (ANNs) are capable of competing with SVMs in a field which was long dominated by SVMs. The combined CNN for all categories naturally scores lower with an average AUC value of 0.82 ± 0.06 . Its great advantage over the individual networks is that it can be applied to future applications of this research much more easily. Whether the benefits of this advantage justify the lower performance is to be decided by the applicant.

The use of LSTM units demonstrated the difficulties which are associated with training RNNs. Their high sensitivity to exactly optimized hyper-parameter makes them unstable compared to the more robust CNN. This was shown by the Standard Deviation (Std) of its performance which was four to six times higher than in the CNN approach for the examined category.

An additional interesting observation of this project is the importance of training good Word2Vec word representations. It was found that most of the network's training effort goes into further adjusting these already pre-trained word vectors. A test run where the vectors are kept static showed that the size of the actual network had to be increased significantly. Proposals on how to provide already well usable word vectors are discussed in Chapter 7.

7 Outlook

Future work that builds upon this project could take various directions. The main areas to look into will most likely be further research in network architectures such as CNNs and RNNs and an improved word vector representation:

As far as the results in this project show, the most promising network architecture for NLP tasks at the moment is the CNN. It proved to be more stable than the RNN and is therefore easier to implement and optimize. This is particularly important given that the data at hand is somewhat difficult to handle. Within the range of CNNs, further research into the combined categories approach will be of interest. The loss in performance of this network compared to the individual approach can be considered tolerable and could be diminished further by exploring a more complex and deep structure towards deep learning. Especially with respect to the intended applications of this project, an improved combined categories approach would be of interest.

The individual CNNs could be improved by optimizing each network not just on regularization but also on the entire architecture. The fact that the one example category which the general architecture for all categories was determined from scored highest gives reasonable hope that individual architectures can lead to general improvement. In the approach with RNNs the most important concern for future work will be the instability. Measures to render the network more robust and ways to obtain better hyper-parameters will need to be found.

In the future, improving the performance of the Word2Vec representation with the data of this project will be challenging but essential. Since the very small corpus makes it hard to train the vectors from scratch we suggest using the pre-trained vectors that the authors of Word2Vec themselves offer together with Google [52]. The disadvantage of this method is then that many of the words in this highly specialized vocabulary are missing in the vocabulary of the Google vectors. Procedure of how to handle these missing words may include random initialization or the complete deletion of these special and rare words. Since the field of NLP in medical applications is growing very fast, there is reasonable hope that specialized vocabularies of pre-trained word vectors for medical topics might be published in the near future as well.

Another interesting approach to tackling the low quality of the Word2Vec vectors is to try other types of vector representations. The GloVe embedding is probably the most promising alternative for word embeddings compared to Word2Vec. Sentence2Vec or Doc2Vec, however, could also yield interesting results when the medical abstracts are not examined on a word level but rather on a sentence or entire abstract level. This approach would then exploit the context between sentences as well as the context within a sentence.

Finally, the problem of having a small text corpus could be tackled by various data augmentation techniques. While this is already common use in image recognition tasks, it is not as sophisticated in NLP yet. Our proposals for such techniques include substituting words in the corpus with synonyms and/or changing the order of phrases within sentences.

A Appendices

A.1 Comparison of optimization methods

There are various more or less sophisticated methods and algorithms to compute the gradients and minimize the error described in Chapter 3.1. Initially, the ADAM optimizer was chosen for all the networks because it is the most commonly used algorithm in the machine learning community. However, the training process of the LSTM networks in the extension of this project showed unforeseen difficulties when trying to minimize the training error with ADAM. Therefore a comparison of alternative optimization algorithms for the LSTM networks was conducted to find a more stable and less fluctuating loss decrease. The three compared algorithms are: Stochastic Gradient Descent (SGD), ADAGRAD and ADAM. This resulted in the use of ADAM for the CNNs and ADAGRAD for the LSTM units.

A.1.1 Stochastic Gradient Descent

The Stochastic Gradient Descent (SGD) algorithm is the most basic optimization method in this comparison and was discussed in Chapter 3.1. The weights \mathbf{w} are updated via the gradient $g = \frac{\partial E}{\partial \mathbf{w}}$ which is multiplied by the learning rate η . One can further add a momentum term v weighted with a parameter γ so also the past gradients are taken into account and the new weight update is less stochastic:

$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta \mathbf{g}_t \quad (\text{A.9})$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{v}_t \quad (\text{A.10})$$

The momentum term increases the weight update if the new gradient points in the same direction and it decreases the update if the direction changes. This way convergence is sped up and oscillations around a minimum are damped. Note however, that the learning rate η and the momentum parameter γ are each the same for all weights and gradients.

A.1.2 AdaGrad optimizer

Having the same learning rate for all weights and gradients is an easy but somewhat inept approach. The ADAGRAD optimizer [61] (Adaptive Gradient) treats each weight individually by assigning different learning rates which depend on the importance of the particular weight. If the gradients for a weight have been large in the past then the learning rate will be reduced and vice versa. This is particularly helpful when dealing with sparse data as is the case in many NLP tasks. Infrequent words need to be updated in larger steps than frequent words which is why also the GloVe embeddings, see Chapter 3.3.2, were trained using the ADEGRAD optimizer [54]. The update rule for each individual weight \mathbf{w}_i is thus:

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{\sum_t g_{t,i}^2 + \epsilon}} g_{t,i} \quad (\text{A.11})$$

Here, ϵ is a small number that prevents the algorithm from dividing by zero. Since the learning rate is divided by the square root of the sum of the square of all past gradients, the learning of infrequent parameters goes much faster than for frequent ones. The main problem of ADAGRAD however, is that the learning eventually stops for frequent and unfrequent weights alike after a certain number of iterations because there is nothing to stop the sum of gradients to grow.

A.1.3 Adam optimizer

The ADAM optimizer [62] (Adaptive Moment Estimation) combines the ideas from the previously described methods and improves them. It uses both the sum of the gradients (first moment, momentum term) as well as the sum of the square of the gradients (second moment). Both moments are multiplied by exponentially decaying parameters β_1 and β_2 to prevent them from growing too much:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t \quad (\text{A.12})$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \mathbf{g}_t^2 \quad (\text{A.13})$$

The authors argue that the two moments are biased towards zero especially in the initial steps when the decay rates are small ($\beta \rightarrow 1$). In order to equalize these biases, they define the unbiased moments as:

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t} \quad (\text{A.14})$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (\text{A.15})$$

The update rule for the ADAM optimizer is then:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{\mathbf{m}}_t \quad (\text{A.16})$$

A.1.4 Result of comparison of optimizers

The three optimizers Stochastic Gradient Descent (SGD), ADAGRAD and ADAM were compared for an exemplary LSTM network on category "Methods". The respective learning rates were adapted for each optimizer so the training loss would reach zero within 500 epochs which resulted in learning rates of 0.05, 0.001 and 0.00005 respectively. A clipnorm of 1.0 was applied to all the optimizers and the ADAM optimizer used a decay parameter

of size $(learning\ rate)/(number\ of\ epochs)$. The results can be seen in Figure 26. The loss decrease is by far the most stable with ADAGRAD while it has irregularities for the other two optimizers. The instability of the loss optimization, especially for ADAM, can be seen for the training data and even more emphasized on the validation data. Therefore, the ADAGRAD optimizer was picked for the experiments with RNNs.

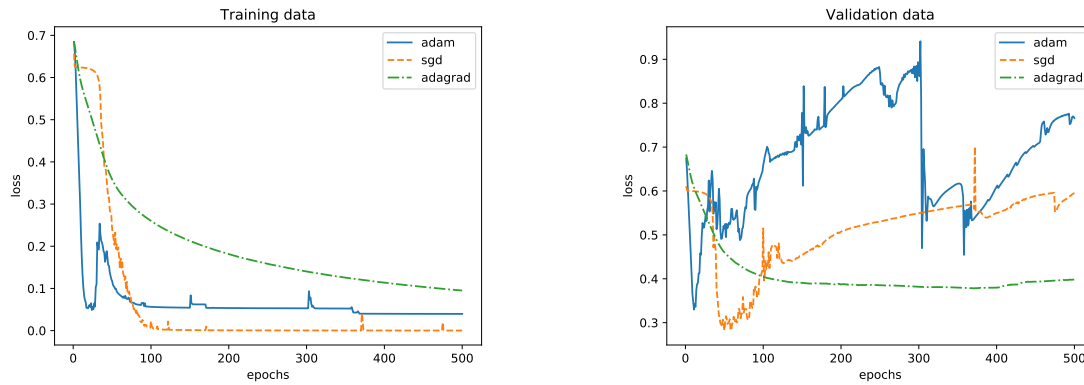


Figure 26: Comparison of the training (left) and validation (right) loss development for the three optimizers.

A.2 CNN individual categories

A.2.1 Network size

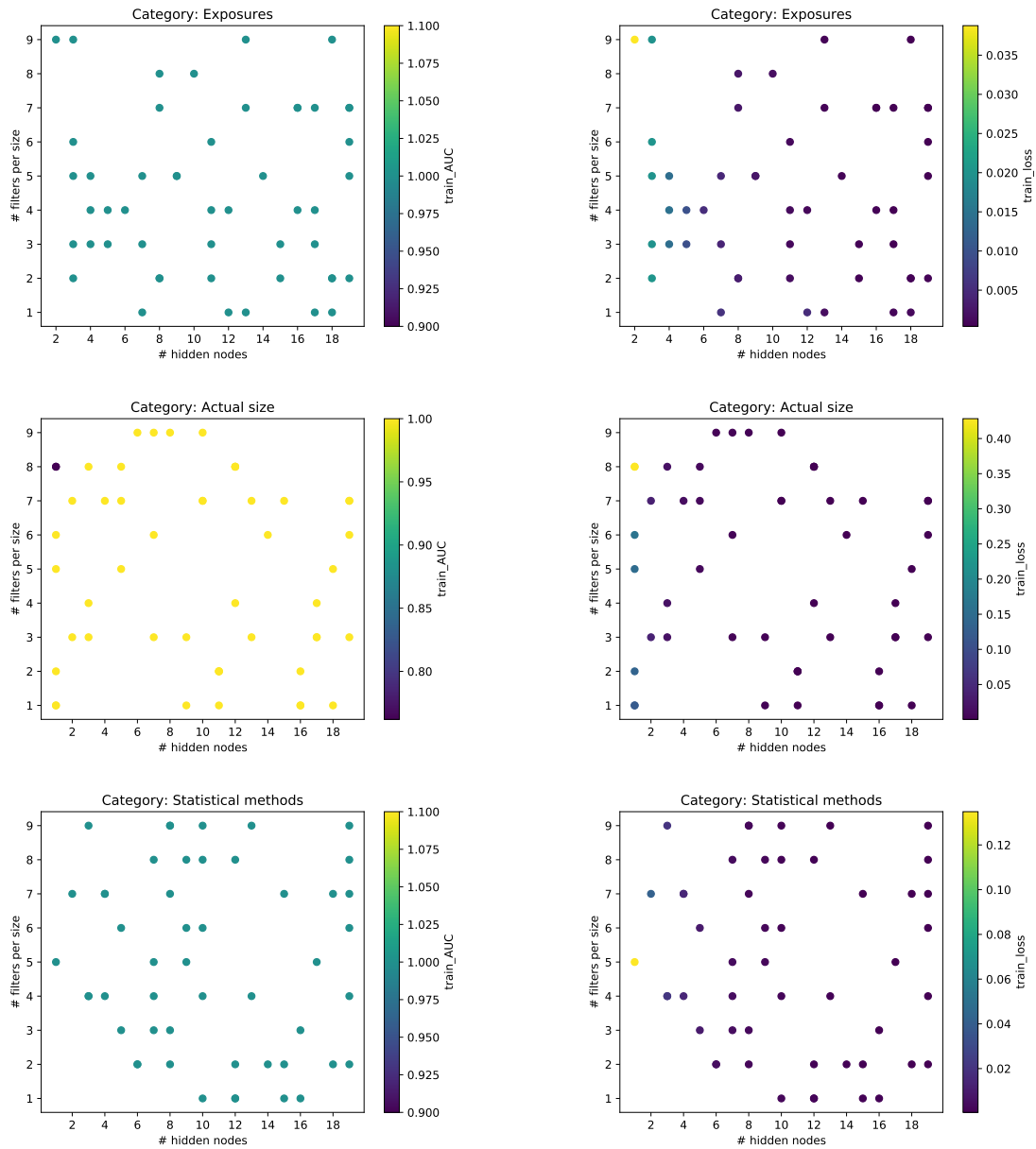
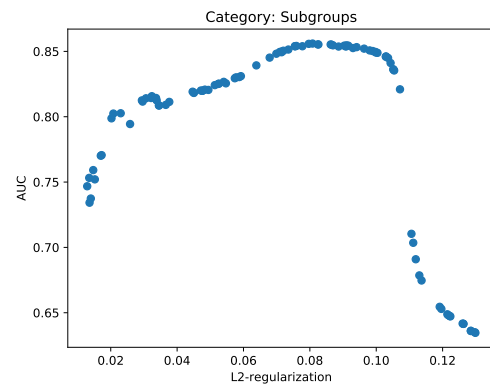
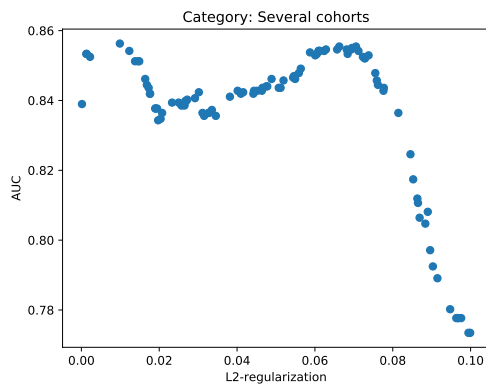
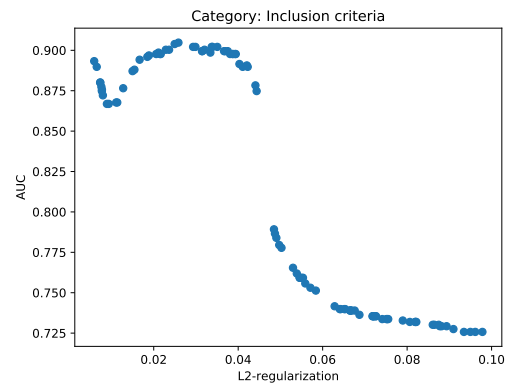
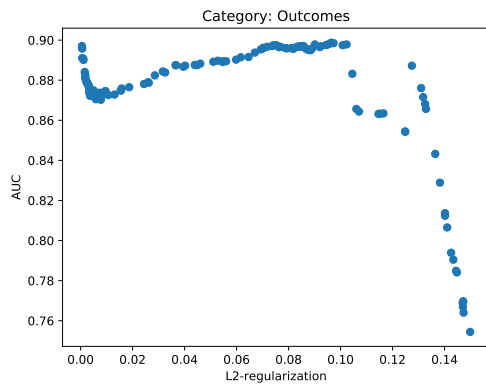
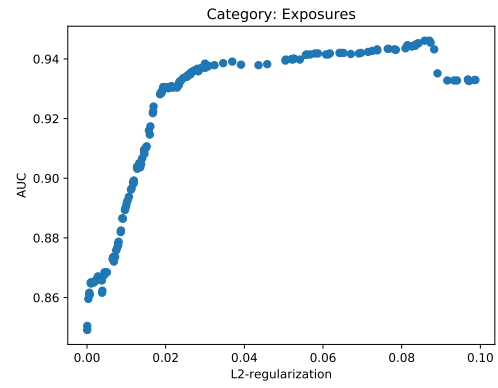
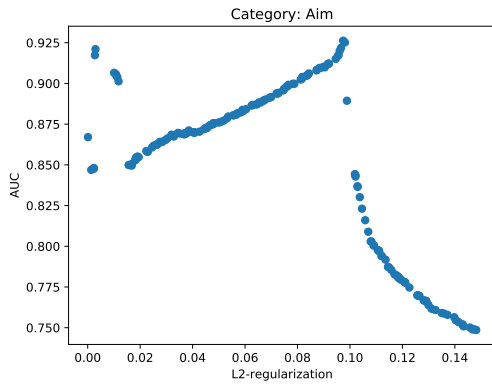


Figure 27: Three example categories which are more easy to categorize than "Study design" in Chapter 5.1.1. The necessary network size to reach a training loss of approximately zero or an AUC of one is smaller than for "Study design".

A.2.2 Regularization



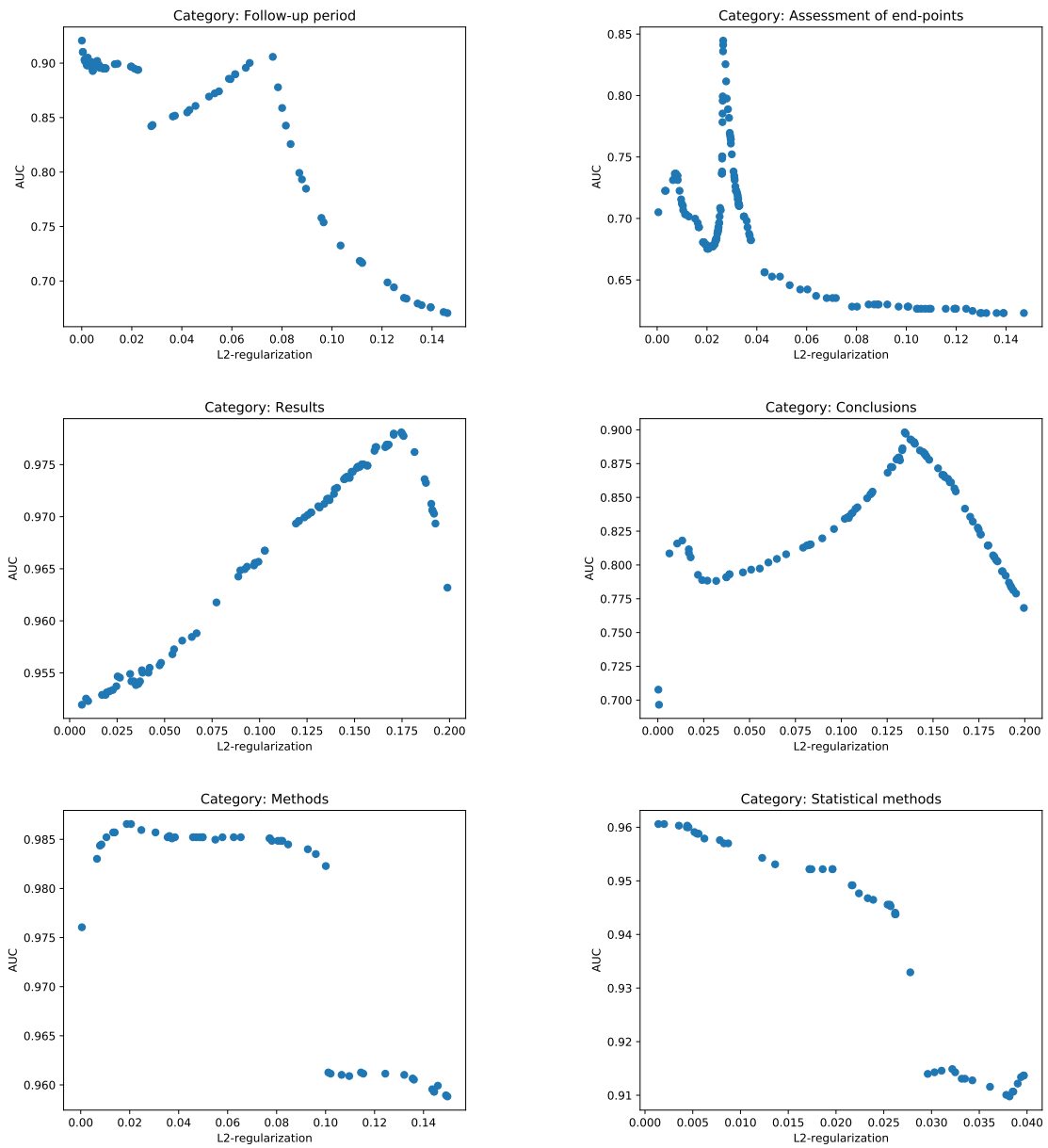


Figure 28: Optimization of each individual CNN from Chapter 5.1.1 on L2-regularization.

A.2.3 Loss-AUC curves

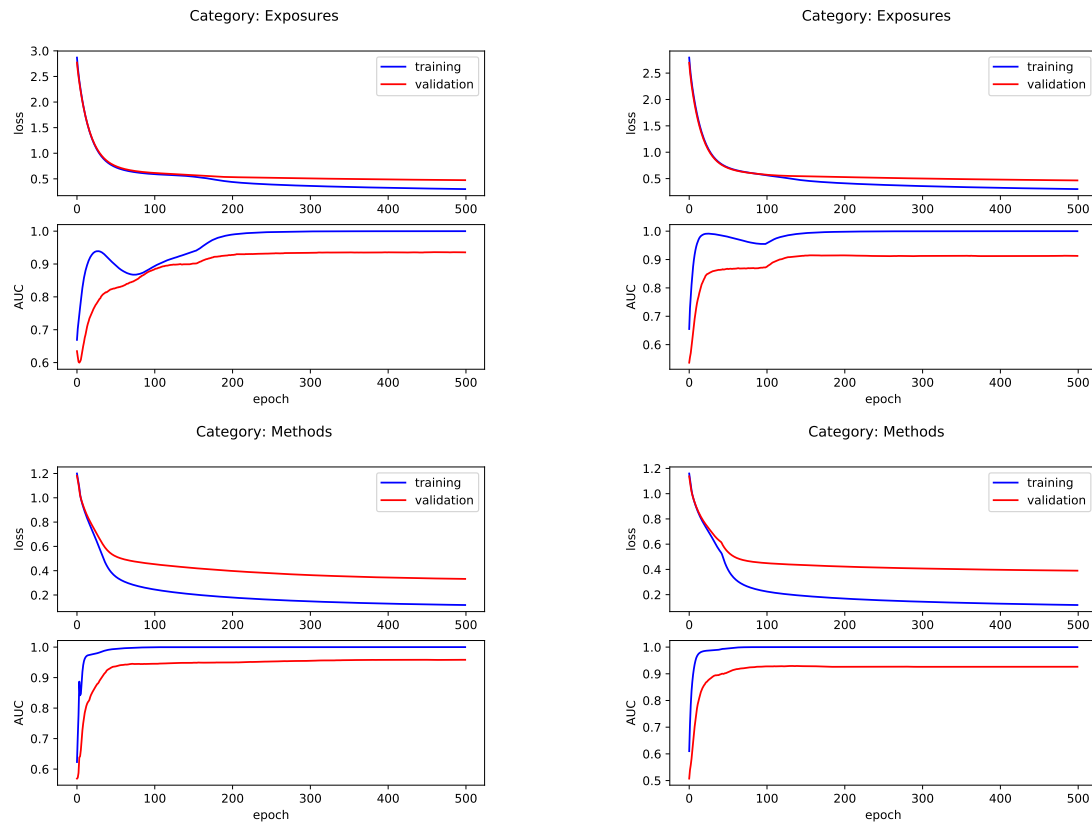


Figure 29: Loss and AUC curves for examples of more frequent categories: Exposures (788) and Methods (313). Their respective K-fold iterations with best (left) and worst (right) results agree well.

A.3 CNN combined categories

A.3.1 Regularization

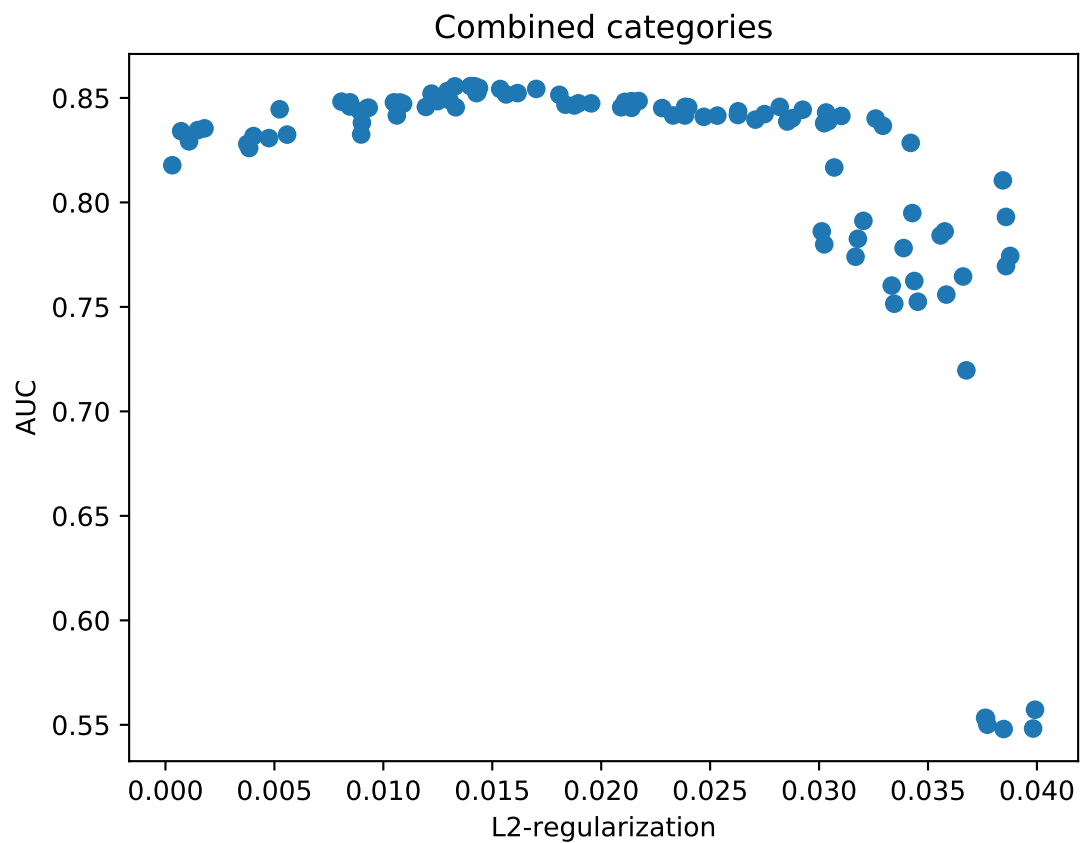


Figure 30: L2-regularization for the combined network. The best validation AUC score is reached for a regularization of 0.014.

A.4 Software

Library: Python
Version: 3.6.2
License: GPL-compatible
Homepage: <https://www.python.org/>
Install date: 18th December 2017

Library: Tensorflow
Version: 1.3.0
License: Apache 2.0
Homepage: <https://www.tensorflow.org/>
Install date: 18th December 2017

Library: Keras
Version: 2.1.2
License: MIT
Homepage: <https://keras.io/>
Install date: 18th December 2017

Acronyms

ANN Artificial Neural Network. 2, 3, 4, 14, 15, 16, 18, 19, 22, 24, 27, 28, 34, 36, 37, 52

AUC Area Under the Curve. 11, 38, 40, 41, 42, 43, 45, 46, 47, 48, 49, 50, 52

CBOW Continuous Bag-Of-Words. 30, 31

CNN Convolutional Neural Network. 1, 2, 3, 4, 5, 10, 14, 15, 19, 20, 21, 22, 23, 24, 36, 38, 39, 40, 41, 46, 49, 51, 52, 53, 58

EPIC European Prospective Investigation into Cancer and Nutrition. 6

FPR false-positive rate. 11

LSA Latent Semantic Analysis. 30

LSTM Long Short-Term Memory. 5, 24, 25, 26, 39, 40, 41, 49, 50, 51, 52, 54, 55

MDC Malmö Diet and Cancer. 6

NLP Natural Language Processing. 1, 2, 3, 4, 5, 14, 19, 20, 22, 23, 24, 32, 35, 36, 50, 51, 52, 53, 54

ReLU Rectified Linear unit. 18

RNN Recurrent Neural Network. 1, 3, 4, 5, 10, 14, 15, 24, 41, 49, 50, 52, 53, 55

ROC Receiver Operating Characteristic. 11

SGD Stochastic Gradient Descent. 18, 54

StD Standard Deviation. 40, 41, 52

SVM Support Vector Machine. 1, 4, 14, 33, 34, 35, 36, 40, 42, 43, 46, 52

TPR true-positive rate. 11

References

- [1] Kurdi, Mohamed Zakaria. *Natural Language Processing and Computational Linguistics 2: Semantics, Discourse and Applications*. Vol. 2. John Wiley and Sons, 2017.
- [2] Hutchins, W. John. "The Georgetown-IBM experiment demonstrated in January 1954." *Conference of the Association for Machine Translation in the Americas*. Springer, Berlin, Heidelberg, 2004.
- [3] Johnson, Mark. "How the statistical revolution changes (computational) linguistics." *Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?*. Association for Computational Linguistics, 2009.
- [4] Bornmann, Lutz, and Rüdiger Mutz. "Growth rates of modern science: A bibliometric analysis based on the number of publications and cited references." *Journal of the Association for Information Science and Technology* 66.11 (2015): 2215-2222.
- [5] Kim, Yoon. "Convolutional neural networks for sentence classification." *arXiv preprint arXiv:1408.5882* (2014).
- [6] Johan Frid et al.
- [7] Zhang, Ye, and Byron Wallace. "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification." *arXiv preprint arXiv:1510.03820* (2015).
- [8] Kalchbrenner, Nal, Edward Grefenstette, and Phil Blunsom. "A convolutional neural network for modelling sentences." *arXiv preprint arXiv:1404.2188* (2014).
- [9] Wang, Peng, et al. "Semantic clustering and convolutional neural network for short text categorization." *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Vol. 2. 2015.
- [10] Johnson, Rie, and Tong Zhang. "Effective use of word order for text categorization with convolutional neural networks." *arXiv preprint arXiv:1412.1058* (2014).
- [11] Johnson, Rie, and Tong Zhang. "Semi-supervised convolutional neural networks for text categorization via region embedding." *Advances in neural information processing systems*. 2015.
- [12] Nguyen, Thien Huu, and Ralph Grishman. "Relation extraction: Perspective from convolutional neural networks." *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*. 2015.

-
- [13] Sun, Yaming, et al. "Modeling Mention, Context and Entity with Neural Networks for Entity Disambiguation." IJCAI. 2015.
- [14] Zeng, Daojian, et al. "Relation classification via convolutional deep neural network." Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers. 2014.
- [15] Gao, Jianfeng, et al. "Modeling interestingness with deep neural networks." Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014.
- [16] Shen, Yelong, et al. "A latent semantic model with convolutional-pooling structure for information retrieval." Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management. ACM, 2014.
- [17] Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." International Conference on Machine Learning. 2014.
- [18] Zhang, Xiang, Junbo Zhao, and Yann LeCun. "Character-level convolutional networks for text classification." Advances in neural information processing systems. 2015.
- [19] Zhang, Xiang, and Yann LeCun. "Text understanding from scratch." arXiv preprint arXiv:1502.01710 (2015).
- [20] Tai, Kai Sheng, Richard Socher, and Christopher D. Manning. "Improved semantic representations from tree-structured long short-term memory networks." arXiv preprint arXiv:1503.00075 (2015).
- [21] Yin, Wenpeng, et al. "Comparative study of cnn and rnn for natural language processing." arXiv preprint arXiv:1702.01923 (2017).
- [22] Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling." arXiv preprint arXiv:1803.01271 (2018).
- [23] Manjer J, et al. The Malmö Diet and Cancer Study: representativity, cancer incidence and mortality in participants and non-participants. Eur J Cancer Prev. 2001;10:489-99.
- [24] Hamrefors V, Hedblad B, Hindy G, Smith JG, Almgren P, Engström G, et al. (2014) Smoking Modifies the Associated Increased Risk of Future Cardiovascular Disease by Genetic Variation on Chromosome 9p21. PLoS ONE9(1): e85893.
- [25] Barregard L, Sallsten G, Fagerberg B, Borné Y, Persson M, Hedblad B et al. Blood Cadmium Levels and Incident Cardiovascular Events during Follow-up in a Population-Based Cohort of Swedish Adults: The Malmö Diet and Cancer Study. Environ Health Perspect 2016;124:594–600.

- [26] Riboli E. Nutrition and cancer: background and rationale of the European Prospective Investigation into Cancer and Nutrition (EPIC). *Ann Oncol* 1992;3:783-91.
- [27] Agarwal S, Yu H. Automatically classifying sentences in full-text biomedical articles into introduction, methods, results and discussion. *Bioinformatics* 2009, 25(23):3174–3180.
- [28] Simon Haykin. 1998. *Neural Networks: A Comprehensive Foundation* (2nd ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA.
- [29] Ohlsson, Mattias. Lecture notes: "Introduction to Artificial Neural Networks and Deep Learning" (2018)
- [30] Sonoda, Sho, and Noboru Murata. "Neural network with unbounded activation functions is universal approximator." arXiv preprint arXiv:1505.03654 (2015).
- [31] Karlik, Bekir, and A. Vehbi Olgac. "Performance analysis of various activation functions in generalized MLP architectures of neural networks." *International Journal of Artificial Intelligence and Expert Systems* 1.4 (2011): 111-122.
- [32] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.
- [33] Saxena, Abhineet. "Convolutional Neural Networks (CNNs): An Illustrated Explanation." XRDS, 29 June 2016, xrds.acm.org/blog/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/.
- [34] Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015
- [35] <http://cs231n.github.io/convolutional-networks/#pool>
- [36] Britz, Denny. "Understanding Convolutional Neural Networks for NLP." *WildML*, 10 Jan. 2016, www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/.
- [37] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems* 27, pp. 3320–3328. Curran Associates, Inc., December 2014.
- [38] Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." *European conference on computer vision*. Springer, Cham, 2014.
- [39] Zeiler M.D., Fergus R. (2014) *Visualizing and Understanding Convolutional Networks*. In: Fleet D., Pajdla T., Schiele B., Tuytelaars T. (eds) *Computer Vision – ECCV 2014*. ECCV 2014. *Lecture Notes in Computer Science*, vol 8689. Springer, Cham

- [40] Elman, Jeffrey L. "Finding structure in time." *Cognitive science* 14.2 (1990): 179-211.
- [41] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.
- [42] Olah, Christopher. "Understanding LSTM Networks." *Understanding LSTM Networks – Olah's Blog*, 27 Aug. 2015, colah.github.io/posts/2015-08-Understanding-LSTMs/.
- [43] Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research* 15.1 (2014): 1929-1958.
- [44] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." *Machine learning* 20.3 (1995): 273-297.
- [45] Kowalczyk, Alexandre. *Support Vector Machines Succinctly*. Syncfusion, 2017, www.svm-tutorial.com/.
- [46] Upadhyaya, Shruti, and R. A. A. J. Ramsankaran. "Support Vector Machine (SVM) based Rain Area Detection from Kalpana-1 Satellite Data." *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 2.8 (2014): 21.
- [47] Moreira, Catarina. *Learning to rank academic experts*. Diss. Master Thesis, Technical University of Lisbon, 2011.
- [48] Larson, Ray R. "Introduction to information retrieval." *Journal of the American Society for Information Science and Technology* 61.4 (2010): 852-853.
- [49] Zanaty, E. A. "Support vector machines (SVMs) versus multilayer perception (MLP) in data classification." *Egyptian Informatics Journal* 13.3 (2012): 177-183.
- [50] Osowski, Stanislaw, Krzysztof Siwek, and Tomasz Markiewicz. "Mlp and svm networks-a comparative study." *Signal Processing Symposium, 2004. NORSIG 2004. Proceedings of the 6th Nordic*. IEEE, 2004.
- [51] Firth, John R. "A synopsis of linguistic theory, 1930-1955." *Studies in linguistic analysis* (1957).
- [52] Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781* (2013).
- [53] Huang, Eric H., et al. "Improving word representations via global context and multiple word prototypes." *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics, 2012.

- [54] Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.
- [55] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.
- [56] Baroni, Marco, Georgiana Dinu, and Germán Kruszewski. "Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors." Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Vol. 1. 2014.
- [57] Levy, Omer, and Yoav Goldberg. "Neural word embedding as implicit matrix factorization." Advances in neural information processing systems. 2014.
- [58] Österlund, Arvid, David Ödling, and Magnus Sahlgren. "Factorization of latent variables in distributional semantic models." Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. 2015.
- [59] Wang, Sida, and Christopher D. Manning. "Baselines and bigrams: Simple, good sentiment and topic classification." Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2. Association for Computational Linguistics, 2012.
- [60] Mesnil, Grégoire, et al. "Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews." arXiv preprint arXiv:1412.5335 (2014).
- [61] Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." Journal of Machine Learning Research 12.Jul (2011): 2121-2159.
- [62] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
- [63] <https://github.com/keras-team/keras/issues/2280>
- [64] <https://keras.io/getting-started/faq/#how-can-i-obtain-reproducible-results-using-keras-during-development>
- [65] Rehurek, Rahim. "Gensim." Software Framework for Topic Modelling with Large Corpora, ELRA, 2010, is.muni.cz/publication/884893/en.
- [66] Chollet, Francois. "Keras." 2.1.2, 2015, keras.io.
- [67] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané,

Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from [tensorflow.org](https://www.tensorflow.org)

- [68] Pedregosa, Fabian, et al. “Receiver Operating Characteristic (ROC).” Receiver Operating Characteristic (ROC) - Scikit-Learn 0.19.1 Documentation, Scikit-Learn, scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html#sphx-glr-auto-examples-model-selection-plot-roc-py.