Multi-level Deep Models for Forum Sentiment Classification

Master's Thesis

Peter Fogh
Emil A. L. Nesgaard

Aalborg University
Department of Computer Science
Selma Lagerlöfsvej 300
DK-9220 Aalborg Øst
http://www.cs.aau.dk

AALBORG UNIVERSITY

STUDENT REPORT

Aalborg University
Department of Computer Science
Selma Lagerlöfsvej 300
DK-9220 Aalborg Øst
http://www.cs.aau.dk

Title:
Multi-level Deep Models for Forum Sentiment Classification

Theme:
Specialization in Machine Intelligence

Project Period:
Spring Semester 2017

Project Group:
mi103f17

Participants:
Peter Fogh
Emil A. L. Nesgaard

Supervisors:
Thomas Dyhre Nielsen
Bo Thiesson

Report Page Numbers: 66

Total Page Numbers: 70

Date of Completion:
June 9, 2017

Abstract:

Multiple attempts have been made to address the problem of target dependent sentiment classification, however it is mostly done for tweets. Another interesting domain is that of forum threads. Analysing forums with cancer patients' described effects of supplementation for their treatment, could be beneficial for other patients looking to improve their situation. To solve this problem we propose three LSTM-based methods, which models the natural hierarchy of forum threads, by considering both word and post-level contexts with regards to targets. Each model captures this hierarchy differently, exploring the relation of word and document embeddings, and how multiple labels and loss functions can improve performance. We manually annotate threads from various cancer-related forums for evaluating our models. Our results show potential in our models designs, and that given more data, they should be able to outperform previous models.

Preface

This report has been developed by 10th semester Software Engineering students at Aalborg University, during the spring semester of 2017. It is a master's thesis project within the area of machine intelligence. The project idea and the dataset from the cancer forums was provided by the Danish data analytics company Enversion A/S in collaboration with Bo Thiesson and was supervised by Thomas Dyhre Nielsen and Bo Thiesson.

We would like to thank Enversion A/S for providing the project proposal, as well as the forum dataset used for the project and an idea for a baseline algorithm to compare our methods to. We would also like to thank Thomas Dyhre Nielsen and Bo Thiesson for their excellent supervision throughout the project.

References and citations are done by the use of numeric notation. For instance [1] refers to the first item in the bibliography. Figures, Tables, Equations, and the like are named with a number according to the chapter they are represented in. As an example, the second figure in Chapter 3 will have the name Figure 3.2.

Aalborg University, June 9, 2017

Emil A. L. Nesgaard
<enesga12@student.aau.dk>

Peter Fogh
<pfogh12@student.aau.dk>

Summary

In this thesis we present the problem of target dependent sentiment classification for online forum threads. Forums are interesting to analyse for sentiment classification, as they are rarely self-contained, but instead are part of a larger context, where they can refer to things previously described by preceding posts in the same thread, or asking questions which are answered in following posts. This observation shows that forum posts have a sequential nature across a thread, in which the targets we wish to classify the sentiment of, can take part in a discussion spanning multiple posts. At the same time, the text inside the posts is also sequential by nature, as the semantics of sentences depends on the order of words. As such a forum thread can be viewed as a hierarchical structure of sequential data.

The goal of this thesis is to model this hierarchical structure using deep neural networks and show that considering continuous, multi-level contexts can improve target dependent sentiment classification accuracy over different baseline methods. To do this, we propose three Long Short-Term Memory-based models, which each models the problem differently, exploring the relation and combination of word and document feature vector embeddings, and how utilizing multiple class labels and loss functions can affect the performance.

We evaluate our models on data from various online cancer forums. More specifically we focus on the topic of cancer patients' experiences with supplementations and their effects on cancer treatment, as this is a widely discussed topic across the forums. As we use supervised learning models we manually annotate forum threads with positive, negative or neutral labels, to use as multiclass training and test data for our models. We have created an annotating strategy and developed a command-line tool for annotating more efficiently.

When evaluating our models on a validation set, we see that our models have potential and outperform the various baselines. However, when evaluated on the test set, performance is decreased significantly. We conclude that the annotated training set is not representative, making it hard for all the evaluated models to produce meaningful results on the test set. We show that by simplifying the problem to a binary classification problem, performance is increased, but to ultimately evaluate the model designs, more training data is needed.

Contents

1 Introduction **1**
1.1 Problem Analysis 3

2 Related Work **9**
2.1 Recurrent Neural Networks 9
2.2 Document Sentiment Classification 9
2.3 Target Dependent Sentiment Classification 10

3 Data Preparation **13**
3.1 Data Description 13
3.2 Manually Annotated Data 15

4 Models **21**
4.1 Notation 22
4.2 Neural Networks 22
4.3 Embeddings 25
4.4 Recurrent Neural Networks 30
4.5 Our Models 37

5 Experiments **45**
5.1 Experimental Data and Setting 45
5.2 Models 47
5.3 Results 49

6 Conclusion **57**

7 Future Work **59**
7.1 Additional Training Data 59
7.2 Active Learning 59
7.3 Scaling the Loss Function 60
7.4 Create Better Input Data 60
7.5 Normalizing the Embeddings for Target Synonyms 60

Bibliography **63**

A Data Analysis **67**
A.1 Supplement Occurrences Analysis 67
A.2 Annotator Agreement Study 67
A.3 Experiments 70

Chapter 1

Introduction

When diagnosed with cancer, patients often search the internet for information regarding their diagnosis and treatments. However, the internet contains much more information than one person can process, especially on such a greatly discussed topic as cancer. Thus, patients often get an incomplete or biased knowledge regarding their situation or opinions on treatment. In the worst case, this could result in a patient stopping the evidence-based traditional treatment, and try unproven alternative treatments, which might not work at all. As such, the ability to automatically gather, understand and aggregate the vast amount of patients opinions and experiences, publicly available on the internet, provides numerous interesting and beneficial applications, helpful for newly diagnosed cancer patients. Such an application could also be beneficial for The Danish Cancer Society, as multiple persons call their hotline every day with a wide variety of questions, especially to unproven or unresearched treatments.

In this project, we investigate how sentiment classification of the posts within varies cancer forums can help gather and process such information, to provide a picture of patients' experiences with cancer treatment. A popular alternative or additional treatment to the standard treatment is concerned with supplementation. The benefits of, for instance, dieting, taking additional micro-nutrients or other remedies to improve the overall quality of life are often discussed within the forums, making supplementation an interesting topic to examine.

As such opinions regarding such supplements will be the main focus of our performed sentiment classification. Targeting the sentiment classification towards a specific topic within our domain relates to the field of target dependent sentiment classification [1, 2, 3, 4]. Compared to traditional sentiment classification, which considers the overall sentiment polarity of a document, target dependent sentiment classification is concerned with finding the polarity of a specific target in the document. This task tries to solve the problem that documents often express multiple sentiments of different polarities. For instance, the sentence *"Cannabis*

helped my treatment, but coffee made me sick.", contains two targets *cannabis* and *coffee*. Because one of the targets has a positive polarity and the other has a negative, it is difficult to return a general polarity of the sentence, but when only considering one of the targets, it is more clear.

In previous studies, target dependent sentiment classification has mainly been performed on tweets [1, 2, 3, 4], which are generally shorter text documents than forum posts. Because of the added length of forum posts, the text is even more likely to contain multiple different opinions. Additionally, forum posts are rarely self-contained, but are rather part of a context, as they can implicitly refer to things previously described by another post in the same thread, or asking questions which are answered in following posts. To the best of our knowledge, we are the first to perform target dependent sentiment classification in the context of multi opinionated forum posts.

Neural networks have seen increasing interest the last years, due to their state-for-art performance on many Natural Language Processing (NLP) tasks [5, 6], such as sentiment classification [7, 8]. This project will research the use of recurrent neural networks, more specifically Long Short-Term Memory (LSTM) models, for target dependent sentiment classification on multi-opinionated forums posts. We utilize LSTM models, due to their ability to model the sequential structure of textual data, in contrast to standard feedforward neural networks, which treats each input as an independent instance, making sequential representations infeasible. We propose three deep neural network models, called Flow-LSTM, Split-LSTM and ML-Split-LSTM which performs target dependent analysis by modelling the hierarchical and sequential structures of a forum thread. This is done by utilizing both document and word embedding vectors of the post containing the target phrase, as well the surrounding posts in the forum thread. Thereby, the idea of this hierarchy of sequential data, assumes a target phrase has a local context of words within the post, but also a related global context of surrounding posts in the thread. Modelling both these levels could improve the sentiment classification, in comparison to simply considering words alone.

The project is performed in cooperation with Enversion A/S, as they provide the 6.538.060 utilized cancer forum posts, as well as a baseline algorithm for comparison with our designed models. From this provided dataset we use a small subset for training our models. Because the data consist of unsupervised examples and our models are classifiers utilizing supervised training instances, we have created a manually annotated dataset. This dataset contains 835 posts with their polarity towards the given targets, and additionally, 4743 posts with their general polarity, utilized for the surrounding posts in a thread. We will present the method for this annotation process as well as further preparation of our data in Chapter 3. Afterwards in Chapter 4 we describe the theoretical foundation of our models and explain why our chosen architecture is suitable for this domain, compared to other

similar models. Experiments are then conducted in Chapter 5, with the primary purpose being evaluating our models on a multi-classed target-dependent sentiment classification task. Lastly, we discuss our results and propose changes, alternatives and further improvements to the models. However, at first we formalize our problem and highlight related work in the following section and chapter.

1.1 Problem Analysis

We begin by describing in more detail the fundamental problem of sentiment classification, followed by the more concrete variants of document and target dependent sentiment classification we aim to solve in the context of cancer forum threads. Additionally, we describe our use of machine learning models and the intuition of our modelling of the local and global context of a forum post. Finally, we present our delimitation of the challenges of this project, and formalize our problem in a problem statement.

1.1.1 The Problem of Sentiment Classification

We use Example 1.1 which is a excerpt from a real opinionated cancer forum post, to introduce the problem of sentiment classification.

> **Example 1.1 (Cancer forum post)**
> " ...I personally have reduced the amount of meat I eat while increasing fruits, veggies and grains. I cannot believe the positive differences this change has made in my body. ... "

As seen in Example 1.1, the post expresses both positive and negative polarized opinions due to its comparative relation between meat and fruits, veggies, and grains. Thus, an opinion have a specific target, e.g. negative for meat and positive for veggies. If we consider the most extensive sentiment classification problem, we would need to discover all expressed opinions in this post. Furthermore these opinions are also expressed by an opinion holder at a specific time stamp and so on. For instance, in this post the opinion holder is only the author of the post, and the time can only be assumed to be when the post was posted, but in longer documents there could be many more to find. Overall this is a lot of parameters to consider and discovering and analysing all combinations is a complex NLP task in itself, which means a much simpler sentiment classification problem is often used. One of the most used simplifications in the literature is the document sentiment classification [9]. It simplifies the problem by assuming that only a single opinion is expressed in the post, which is only expressed by a single opinion holder at a

single time. However for less domain-specific documents with multiple targets, this is sort of an abstraction. Therefore, previous studies [1, 2, 3] have presented the problem of target dependent sentiment classification, which is more specific than the document sentiment classification, as it enables discovery of an opinion for a chosen target.

1.1.2 Machine Learning Models

Solving sentiment classification problems can be done using multiple classes of machine learning algorithms. However, it is inherently a classification problem, since a notion of sentiment polarity is hard to extract from the text alone, and thus a supervised algorithm is most suitable for this task. There exist unsupervised algorithms for sentiment classification which achieve good results [10], but supervised methods are generally much better. Especially neural networks and deep learning have proven to be very effective in sentiment classification [8].

However, standard neural networks are limited in which types of data they can process, as their mapping is only from a single fixed-size input vector to output vector. Since we are dealing with textual data, which is sequential in nature, this presents a problem, as the words of a sentence or the post of a thread are not independent on each other. When you read a word, your understanding is based on the previous words in the sentence, and a post which is a response to something, depends on the posts it is preceded by.

To solve this issue we look into recurrent neural networks, and more specifically the Long Short-Term Memory architecture, as these explicitly model the handling of sequential data, by introducing a notion of time steps into the neural network [11]. For these models to process our data, we also need a method of representing the raw text as vectors. There are multiple ways to achieve this, but the embedding methods by Mikolov et al. has proven especially efficient for representing words and documents as feature vectors [5] [12], also specifically for sentiment classification [8].

Other than simply being sequential, there also exists a natural hierarchy in a forum thread, as illustrated in Figure 1.1. A thread is made up of posts, and posts are made up of words. Modelling such a hierarchy could capture the sentiment polarity of a target in a post better than simply considering the words in the post. The intuition is that target dependent sentiment classification can be done for a post containing a target, to get a sentiment polarity for this target, based on the local context of the post. Since forum threads are discussions it is likely that elements outside this local context also influence the sentiment polarity, thus modelling the global context of the thread, meaning the sequence of posts in the thread, can serve as a booster or a normalizer for the sentiment polarity of the target.

Thus, we wish to model the natural hierarchy in a forum thread with regards to sentiment classification. We propose a model applying the two different simpli-

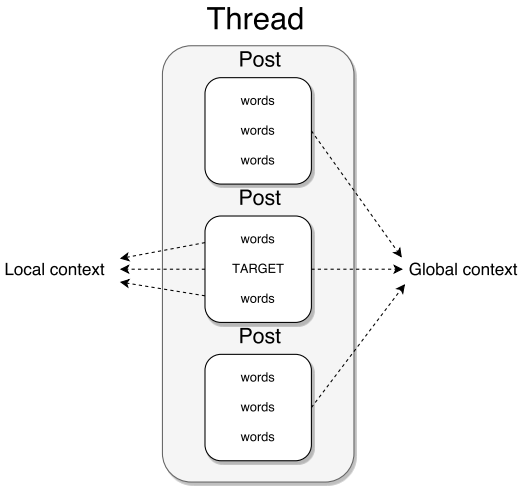


Figure 1.1: The hierarchy of a forum thread.

fied variations of sentiment classification, described in Section **??** into the different levels of the hierarchy, combining them using recurrent neural networks.

1.1.3 The Problem of Target Context Classification

We have established that the challenge of this project is to classify the sentiment of forum posts in regards to supplements. Due to our challenge concerning cancer treatment, we also want to ensure that the supplements used for targets are in this context. Because of our domain of cancer forums, some supplements can be assumed to always relate to cancer, however some more general supplements such as food items or beverages are more likely to be used in other contexts as well, even in cancer forum threads. For instance, the beverage coffee is often mentioned in our forum posts, and as shown in Example **??** it can relate to both cancer and not.

Example 1.2 (Posts Mentioning Coffee)
Coffee in the context of cancer treatment: " ...I should probably cut down on coffee, which is famous for aggravating the lymph nodes and impacting drainage. ... "
Coffee in the a general context: " ... You DO NOT want to see me in the morning without my COFFEE!! ... "

This task can be formulated as a classification problem similar to target dependent sentiment classification. Thus, we utilize the same machine learning models to predict the binary problem of a target being in the context of cancer treatment or not.

1.1.4 Preparing the Data

The dataset provided by Enversion A/S, consisting of threads from various online cancer forums, has been scraped by them and provided in its raw form. As it is a very large amount of textual data from various sources, it is prone to inconsistencies that might cause trouble if used in the models directly. As such, we analyse some of the issues with the raw data and perform a number of different data cleansing steps, to normalize the text and reduce input data related errors.

1.1.5 Training Data

We have chosen to utilize a supervised algorithm, which requires labelled training data. However, as we are dealing with text that has no explicit sentiment polarity labels, we have to manually annotate a set of examples to train our models. To fully capture the semantics of all possible contexts surrounding targets, we should label training examples for all supplements, however this is not feasible due to the amount of possible supplements. Instead we limit the labelling to four different types of supplements and their synonyms, which will serve as the test supplements in our experiments. We will discuss the chosen supplements in Section 3.2.1.

Our goal is to train general models, which can classify the sentiment polarity for any given supplement. However, limiting our training examples to only contain the textual context of four supplements makes the models vulnerable to overfitting these examples during training. There is a risk that the models will only learn to predict for these types of posts and have a hard time for posts with different content. On the other hand, the features for the models primarily consists of the contextual data, not the supplement itself. We will investigate the models ability to generalize to other supplements in our experiments.

In respect to the chosen classification problem of target dependent sentiment classification, we choose to annotate our training instance with three classes for opinion polarities, i.e. positive, negative and neutral. Additionally, we annotate one of the four supplements, which we assume is not always used in the context of cancer, with the boolean label of whether it relates to cancer or not.

1.1.6 Problem Statement

In summary, we have decided to solve the problem of analysing cancer forums, using methods from the domain of target-dependent sentiment analysis. We choose to create supervised learning models, which require us to manually annotate forum posts for the model training. With this previous problem analysis in mind we state our problem as follows:

Problem Statement: *How can the hierarchical and sequential structures of a forum thread be modelled in a target dependent way using Long Short-term Memory networks?*

To be able to answer the problem statement, the following research questions should be answered as well:

- *What are the challenges of manually annotating forum threads?*
- *How can word and document embedding vectors be successfully combined as input to one model?*
- *How do we optimize the models with the purpose of target dependent sentiment classification?*
- *How can document sentiment labels be incorporated as a normalizer?*

Chapter 2

Related Work

This work is connected to multiple areas of NLP, each with their own large amount of related work. We can of course not list the entirety of it, but in this section we outline some of the most prominent work within sentiment classification, both document and target dependent, as well as related deep learning models for these tasks.

2.1 Recurrent Neural Networks

Recurrent neural networks are a logical choice for sentiment classification as it is concerned with processing text, which is inherently sequential. As such there has been much research in using recurrent neural networks for creating language models [13], learning text vector representations [14] and modelling text on different levels, i.e. words and documents, for sentiment classification [15]. Tang, Qin, and Liu learns sentence representations using LSTM networks and afterwards the semantics of sentences and their relations are encoded using recurrent neural networks [14]. Huang, Cao, and Dong also uses LSTM networks, by creating a hierarchy of LSTM networks, combining both the text in a tweet with a representation of the tweet with its replies [15].

2.2 Document Sentiment Classification

Sentiment classification is a fundamental NLP task and have been studied extensively for a long time. It has been done in many different domains, such as product reviews, movie reviews and more recently social media, especially twitter. Some of the most popular data sets for benchmarking algorithms concerns movie reviews and consists of a data set of IMDB reviews and the Stanford Sentiment Treebank, which is based on RottenTomatoes movie reviews.

Traditional document sentiment classification models mostly consists of probabilistic classifiers, as well as statistical methods [16], and the introduction of the IMDB data sets also featured a method based on graph minimum cuts [17]. Since then deep learning models have proven very effective at these tasks and the performance on the IMDB data set has increased considerably [8]. Learning good feature representations of the text has also been the topic of much research, where Paragraph Vector by Le and Mikolov has been shown to yield state-of-the-art results when combined with classifiers such as maximum entropy or multi-layer perceptron [12, 8].

The Stanford Sentiment Treebank was introduced more recently by Socher et al., based on the RottenTomatoes data set introduced by Pang and Lee, to remedy some of the challenges for sentiment compositionality [7]. To address this they also introduce the Recursive Neural Tensor Network which, when trained on this treebank, outperforms previous state-of-the-art methods. Since then even more deep learning models uses this data as a benchmark and further pushes the state-of-the-art [8]. This includes convolutional neural networks [19] and LSTM networks [20] among others.

2.3 Target Dependent Sentiment Classification

The previous mentioned methods all perform very well for the sentiment classification task, achieving results around 90% [8]. However, those results are only for movie reviews. Sentiment classification on shorter length text such as tweets has shown to be a much more difficult task, as they provide less overall signal per document [21]. These methods also have in common that they perform document sentiment classification.

For the case of tweets, a study by Jiang et al. [1] indicates that up to 40% of their errors regarding target dependent sentiment classification, were because the target is not taken into account. This has inspired continuous research within the area of target dependent sentiment classification. Jiang et al. propose a context based approach, which uses related tweets to improve the sentiment classification. They use hand crafted lexical rules focusing on the target and surrounding Part Of Speech (POS) tagged words, combined with *Support Vector Machine*s (SVMs) for classification.

Dong et al. [2] were the first to introduces a public available annotated data set for target dependent twitter sentiment classification, They proposes AdaRNN, an adaptive recursive neural network, which adaptively propagates the sentiments of words to targets depending on the context and syntactic relations.

In the field of LSTM networks, Tang et al. explored the used of LSTM networks processing the text of a tweet from both directions, on each side of the target words [4]. They argue that a standard LSTM solves target dependent sentiment

classification in a target independent way, since the feature representation used for sentiment classification remains the same without taking the target words into account. As a response they introduce two LSTM models, Target-Dependent LSTM (TD-LSTM) and Target-Connection LSTM (TC-LSTM), which both model the preceding and following contexts surrounding the target string, so that contexts in both directions can be used as feature representations for sentiment classification and then combined into a target dependent sentiment prediction. Their proposed TC-LSTM model shows state-of-the-art performance, with an accuracy of 71.5% on the target dependent twitter sentiment data set, introduced by Dong et al. [2].

Subsequently, Ghosh et al. demonstrated that incorporating contextual features, such as targets, into standard LSTM networks can be beneficial for various NLP tasks [6]. Their idea is to adapt the standard LSTM cells into contextual LSTM cells, by modifying the equations for the operations in the cells, adding an embedding vector of target words to each gate.

Chapter 3

Data Preparation

The main data for this project consist of forum threads scraped from various online forums. This data was initially scraped and provided by Enversion A/S, but before we can process it and perform sentiment classification, we need to prepare the data for this task. Thus, in this chapter we describe the data used for the project and how we have prepared it for experiments. This includes various cleansing steps, to normalize the large amount of text and to fix potential scraping errors. Furthermore, to create training data for our models, we manually annotate parts of the dataset, using an annotating strategy and a command-line tool we have developed for this task. We present this dataset, how it is created, as well as the limitations, challenges and potential issues with this approach.

3.1 Data Description

The provided data consists of two datasets containing cancer forum threads and information regarding various supplements. The forum dataset is scraped from five different major cancer related forums, and as shown in Table 3.1, includes a total of 472.758 threads containing 6.538.060 posts. These numbers are based on initial pre-processing, where invalid posts have been removed. This includes duplicate posts and posts which presents issues for how we wish to model the thread. Some posts have no date, which makes it impossible to consider the order of the posts in the thread. Since this is essential for our model design, we choose to discard the threads containing such posts from our dataset.

This tables shows that the bulk of the dataset is made up of threads from breastcancer.org and inspire.com. Inspire is not a forum exclusive to cancer, but contains multiple health communities. However, the scraping was restricted by Enversion A/S to only consider cancer related communities.

The dataset of supplements is scraped from rxlist.com. RxList provides content written by physicians and data provided by credible sources like the U.S. Food and

Table 3.1: Cancer related forums, and the number of scraped threads and posts.

Forum name	# threads	# posts
breastcancer.org	122.420	2.609.762
cancercompass.com	71.491	422.632
cancerforum.net	37.514	290.699
csn.cancer.org	109.586	1.072.584
inspire.com	131.747	2.142.373
Total	472.758	6.538.060

Drug Administration [22]. The data set contains 1.075 different supplements, each with a list of synonyms, resulting in a total of 18.823 synonyms. These synonyms are written both in medical, scientific terms and also in slang. For instance, the synonyms for cannabis include both the words marijuana and grass.

3.1.1 Data Cleansing

As we have a very large amount of textual data, it is prone to inconsistencies that might cause trouble during processing. As such, we cleanse the textual content of each forum post using the processes described in the following, with the purpose of normalizing the text:

- Repeating punctuations are reduced to only one instances. For example, the string "coffee????" is reduced to "coffee ? ". Notice a space is inserted on each side of the punctuation to ease the task of correct tokenization.
- Repeating spaces are reduced to only one space.
- The character "NO-BREAK SPACE" is replaced with the traditional "SPACE" character.
- Line breaks in the start and the end of a post are removed.
- A space between ".", "?", and "!" and all uppercase letters is inserted, to ensure better tokenization of words. For example, the string "cannabis.I am" is transformed to "cannabis. I am". This problem mostly occurs, because some line breaks of the original data have not been captured by the scraper, resulting in concatenated words.
- Forum dependent strings such as "** Originally posted by [username] **" and "01/01/2010 - edited by [username]"' are removed if they occur in the start of the post.

- Images embedded in the post using HTML tags are replaced by the token "[IMAGE]". All other embedded HTML tags such as "<a ... >" or "< br>" are removed.
- URLs, i.e tokens starting with "http://" or "www." are replaced by the common token "[URl]"
- Dates written in the format "01/01/2004" or variances there of are replaced by the common token "[DATE]".

3.2 Manually Annotated Data

We now present how we manually annotate data examples used for training and evaluating our models. As presented in Section 1.1, we investigate the use of both general polarity labels and target dependent polarity labels, from now on referred to as general labels and target labels. We annotate both these types of labels using three classes, positive, negative and neutral. The dataset is annotated on a document level. For posts containing a target, this results in one general label per post and a number of target labels corresponding to the amount of different targets in the post. For the context posts surrounding the target posts, we annotate them using only a general label, as they do not contain any targets.

We also consider the problem of whether a target relates to cancer or not. The intuition of this problem is that some targets have a high probability of appearing in other contexts than ones directly related to cancer and treatment. It would be beneficial to be able to detect this, when attempting to provide the overall polarity of a target, as in cases where the sentiment does not relate to cancer treatment, it should not count towards the overall score. As such, we introduce a third type of label, where the classes are true and false, to capture this relation to cancer. This is however not the main focus of our annotation process and thus we apply some restrictions. Based on our chosen targets, described in the next section, we consider all except one to always relate to cancer, thus only one target will be annotated using this third label.

3.2.1 Supplement Analysis

As described in Section 1.1, we limit our targets to be concerned with supplementation. Supplements are widely discussed across the cancer forums, and with the data provided by Enversion A/S, there is a good foundation for analysing the many supplements that appear across the threads.

The ideal scenario would be to consider all types of supplements, such that we have a broad and representative annotated dataset. However, as shown in the previous section these is over 1000 supplements, and almost 20.000 when including

all their synonyms. Instead we limit the labelling to four different types of supplements and their synonyms. These four supplements are chosen based on their frequency and the diversity in their relation to cancer. The frequency criterion is based on the assumption, that frequently mentioned supplements are widely discussed supplements and thus are likely to be strongly related to cancer treatment. The second criterion is to compensate for a limited number of supplements, with the assumption that there exist implicit groups of supplements.

As such, we choose "Cannabis", "Coffee", "Vitamin D", and "Curcumin" as our target supplements. As seen in Table A.1 the numbers indicate, that they are all relevant and often discussed within cancer forums.

Table 3.2: The amount of posts mentioning the chosen supplements using all of their synonyms.

Supplement	**# posts**
Cannabis	79.576
Coffee	44.799
Vitamin D	40.173
Curcumin	24.486

These numbers are based on the number of tokens from tokenization using the *Natural Language ToolKit* (NLTK) word tokenizer. All of the most mentioned supplements and reasons for considering these over the others can be seen in Appendix A. In this list, supplements with noisy synonyms have been removed. Because we lowercase all supplements to facilitate the text processing, it has the side effect that some synonyms become ambiguous. For instance, the medicin "SAMe" becomes a very common word when lowercased, and thus have a very high frequency.

They also relates to cancer treatment in different ways. Vitamin D[1] is essential for our bodies and thus often used as a supplement when its levels are low, which they often are during cancer treatment. It has been the subject of many studies and is often directly recommended by doctors as a supplement for cancer treatment [23]. As this is something you would only take with the purpose of supplementation, it can be considered to belong to a group of "pure" supplements.

Curcumin[2] is a chemical produced by varies plants. It is argued to have healthy properties and is often used as a food additive and as a herbal supplement. It is considered by many to be an effective supplement for cancer treatment and several studies has highlighted its properties for this [24, 25]. It is often mentioned in discussions about diets and what to eat during cancer treatment, and can thus be considered a diet or food supplement.

[1] `http://www.rxlist.com/vitamin_d/supplements.htm`
[2] `http://www.rxlist.com/turmeric/supplements.htm`

Medical cannabis[3] has been the topic of many discussions, because of the laws regarding its general consumption and because many people believe it has healthy properties for cancer treatment. However just as many people believe the opposite, and multiple papers has been published about the dangers of cannabis [26].

The last chosen supplement, coffee[4], is a not really a supplement to the same degree as the others. It is a very common drink for general consumption and it is thus very likely to not always relate to cancer.

Based on this, coffee is a good candidate to test the relatedness to cancer. We assume that coffee is the only supplement that will often relate to other things than cancer treatment on the given forums, and as a result the relation label is only annotated for posts mentioning coffee. In the following sections the four chosen supplements will be referred to as targets.

3.2.2 Labelling Tool

Each post in our labelled dataset is annotated using a tool developed by ourselves. The tool detects whether a target or one of its synonyms is mentioned in a post. This is done using by tokenizing the words of the post, using the word tokenizer from NLTK and then comparing each token to each synonym. To ensure the detected synonyms are actually mentions of the target, we have excluded all ambiguous synonyms, such as "grass" or "joint" for cannabis. Furthermore, in addition to the synonyms from ReList, we have extended the list of synonyms for each target, with the synonyms found in WordNet [27] and other synonyms found during the labelling of the forum posts, which consists of abbreviations and alternative spellings.

After the labelling tool has detected the posts mentioning one of the targets, the tool randomly select a thread containing such a target post. The annotator is then presented with the first post of the thread, to understand the initial topic of the thread, and must then annotate the general polarity of the post, as well as an additional label for each target mentioned in the post. The tool then continues in the chronological order of the posts in the thread. A screenshot of the tool can be seen in Figure 3.1

Target Dependent Labelling Challenges

We have decided the annotator only has to read the five posts preceding and following each target post. Excluding breastcancer.org, which tends to have very long threads, from the calculation, the average length of a thread is 12 posts. This means that by considering five posts on each side of the target posts, we cover the length

[3] `http://www.rxlist.com/vitamin_d/supplements.htm`

[4] `http://www.rxlist.com/coffee/supplements.htm`

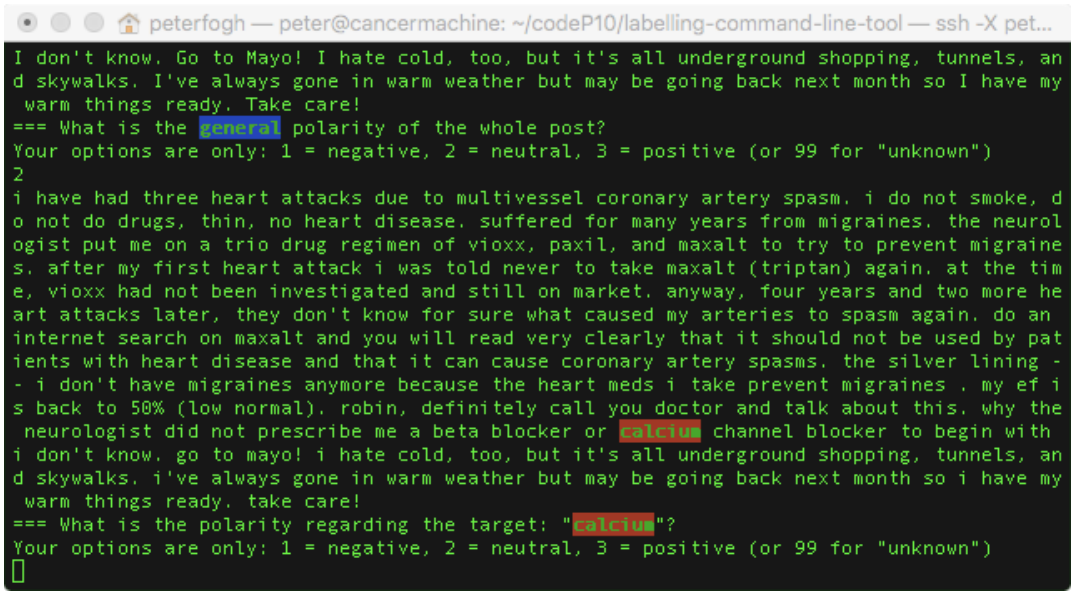


Figure 3.1: Screenshot of the developed labelling tool during labelling of a target.

of most smaller threads. At the same time it is assumed that posts further away from a target mention, will have little relation to the target.

Another challenge to consider is that posts often contains multiple mentions of the same target. We have to keep this in mind when we design our models to be target dependent. The mean number of target occurrences in target posts is $1,62$. This indicates that most target posts only mentions targets a single time and it is not a large problem, but we will keep it in mind for our design regardless.

3.2.3 Labelled Data Set

The authors of this report are the annotators of the manually labelled data set. The final dataset consists of 4.743 general labels, 835 target labels, and 223 labels for coffee's relatedness to cancer. All these labels are over a total amount of 335 forum threads.

Additionally, the distribution of the three polarity classes of the labelled dataset is 13% negative, 33% neural, and 54% positive labels, and the distributions of the binary classes for the cancer related labelled dataset is 70% false and 30% true. The distribution of these sets are slightly unbalanced. We will highlight the implications of this when performing our experiments in Chapter 5. For the experiments we also divide the dataset into a train set for training and optimizing the models, and a test set for evaluating them. This will also be described further in Chapter 5.

Annotator Agreement Study

We have performed an inter-annotator agreement study of the first 20 threads labelled by both annotators. As a agreement metric we utilize Cohen's kappa coefficient [28], which produces a value between -1 and $+1$. The value of 0 represents the amount of agreement that can be expected from random chance, and 1 represents perfect agreement between the annotators. Cohen propose the following interpretation of his coefficient: ≤ 0 as no agreement, $(0, 0.20]$ as none to slight, $(0.20, 0.40]$ as fair, $(0.40, 0.60]$ as moderate, $(0.60, 0.80]$ as substantial, and $(0.80, 1]$ as almost perfect agreement [29].

For our study the first 20 threads labelled is equivalent to: 299 general labels, 56 target labels and 14 relation labels, resulting in the coefficients: $\kappa_{general} = 0.42$, $\kappa_{target} = 0.32$, and $\kappa_{relation} = 1$, respectively.

We conclude, it is difficult to reach a good agreement about the document level polarity, since the $\kappa_{general}$ and κ_{target} indicates a moderate and fair agreement, respectively. This level of agreement is a result of multiple opinions often being expressed in a singe post. Additionally, the result of $\kappa_{target} < \kappa_{general}$, contradicts our initial expectation that we would reach higher agreement for target labels than for general labels, based on the fact that only opinions toward the target should be in consideration. Therefore, to investigate the reason for this result, we read the posts with the 22 disagreeing target labels and discussed their challenges regarding the targets, as described in Appendix A.2. This led to a set of rules we will follow for the remaining labelling, to ensure better agreement.

On the other hand, the study shows that it is much easier to reach agreement for whether a supplement relates to cancer or not, as there was a total agreement, although only for 14 examples.

Chapter 4

Models

With sentiment classification as our objective and the large amount of textual data that has to be processed, we need models that can efficiently learn features of the text with respect to the polarity labels. For such a task, neural network and deep learning models has proven to deliver state-of-the art results. We propose three neural network-based models for target dependent sentiment classification, which aims to model the textual hierarchy of a forum thread, by processing the text both on a word and document level. To capture the structure of the text, we utilize recurrent neural networks, more specifically the Long Short-Term Memory architecture, to model the text as a sequence of either words or documents.

Neural network based models have also been proven effective for learning the semantics of varied-length pieces of text and representing them as fixed-length embedding vectors, which are very useful for text classification. We use such methods to model our dataset and use the resulting vectors as input to our models.

The process from raw text to sentiment classification thus utilizes multiple variations of neural networks. To understand the fundamental theory behind the different techniques used, we start by presenting a standard feedforward neural network, which is the foundation our models are based on. We then present *Word Vector* and *Paragraph Vector*, proposed by Mikolov et al., which are more specific variations of neural networks, and the embedding techniques we use for the vector representation of our text. Afterwards, we present some fundamental issues with standard neural networks in regards to target dependent sentiment classification and introduce recurrent neural networks which improve on these issues. Finally, based on this theory we present our proposed models and argue how multi-level contexts can be used to model forum threads with respect to a target object for sentiment classification.

4.1 Notation

The mathematical notation used throughout this chapter can be seen in Table 4.1.

Table 4.1: Mathematical notation used in this report.

Notation	Description
$\mathbf{v}$	Vector $\mathbf{v}$.
v_i	i'th component in vector $\mathbf{v}$, which is a scalar.
$\lvert\mathbf{v}\rvert$	Length of vector $\mathbf{v}$.
$\mathbf{v}^\top$	Vector $\mathbf{a}$ transposed.
$\mathbf{v} \cdot \mathbf{u}$	Dot product of vector $\mathbf{v}$ and $\mathbf{u}$.
$\mathbf{v} \Vert \mathbf{u}$	Concatenation of vector $\mathbf{v}$ and $\mathbf{u}$.
$\mathbf{M}$	Matrix $\mathbf{M}$.
$\mathbf{M}_{ij}$	Component at i'th row and j'th column in matrix $\mathbf{M}$.
s	Scalar s.
V	Constant V.
$s \cdot k$	Product of scaler s and k.
$f(x) \odot g(x)$	Pointwise product of function f and g.
$\exp(x)$	Natural exponential function of x.

4.2 Neural Networks

Neural networks are made up of small processing units, which are connected using directed, weighted edges. These units are activated when input is provided and this activation then spreads through the weighted connections. In the most simple neural network architecture, the feedforward neural network, the units are divided into different layers which are processed sequentially [30], see Figure 4.1.

To train a neural network, pairs of vectors and associated ground-truth class labels are provided as input. In the context of our sentiment analysis task, an input example could be a vector representation of the words in a post and the ground-truth class label is the polarity score for that post. The intuition is that there exists an implied mapping between data examples and their classes, which can be learned by applying the correct weights. When this mapping has been learned for the training data, where the ground-truth class labels are known, the model can then be used to predict the class label for data examples where it is not known. Formally this means that a neural network with a given set of weight values, $\mathbf{W}$, defines a function, f, from an input vector, $\mathbf{x}$, to a class label, y. Given the input vector $\mathbf{x}$, we wish to find the weight configuration f_W such that $f_W(x) = y$

To learn the optimal configuration of these weights, a forward pass through

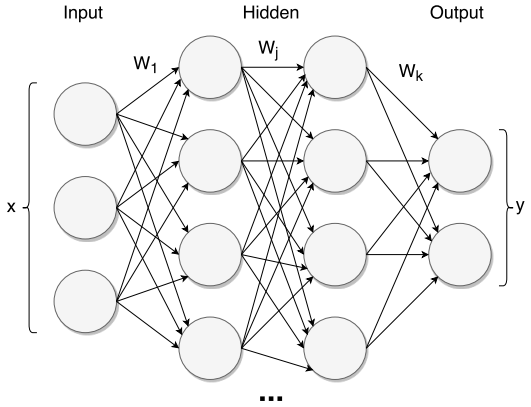


Figure 4.1: A standard feedforward neural network, with single input and output layer and one or multiple hidden layers. The input layer is where data is provided to the network, and the output layer is where a prediction is made. The hidden layer represents a non-linear transformation of the input data, which can happen one or multiple times. This is referred to as shallow and deep neural networks, respectively.

the network is first performed. Formally, in the forward pass each layer j takes an input vector $\mathbf{x_j}$ with size $|\mathbf{x}| = N$ and computes a weighted sum of all values

$$\mathbf{v_j} = \mathbf{W_j x_j} \tag{4.1}$$

A non-linear activation function is then often applied to introduce non-linearity into the network:

$$\mathbf{l_j} = a(\mathbf{v_j}). \tag{4.2}$$

Non-linearity is advantageous for neural networks, as non-linear neural networks are more expressive than linear ones. Neural networks are simply nested functions. If these functions are linear, a neural network with with any number of layers are equivalent to one with a single hidden layer, whereas non-linear function can represent a much larger space of functions [31]. Common choices for non-linear activation functions include the logistic sigmoid function

$$\sigma(x) = \frac{1}{1 + \exp(-\mathrm{x})}, \tag{4.3}$$

and the hyperbolic tangent

$$tanh(x) = \frac{\exp(2\mathrm{x}) - 1}{\exp(2\mathrm{x}) + 1}, \tag{4.4}$$

as well as the Rectified Linear Unit (ReLU)

$$r(x) = \max(0, x). \tag{4.5}$$

One of their common properties, is that they squash their input into a defined range of values. Sigmoid outputs values between 0 and 1, while the hyperbolic tangent outputs values between -1 and 1, which makes them useful for different tasks. ReLU is a bit different, as it is thresholded at 0 and is technically not linear, but rather point-wise linear. If the input domain is restricted to $(-\infty, 0]$ or $[0, \infty)$ it is linear, but it is non-linear across its threshold. This means that it does not fulfill the property of linear functions

$$f(x) + f(y) = f(x + y) \tag{4.6}$$

as

$$f(1) + f(-1) \neq f(0) \tag{4.7}$$

This makes it equivalent to both sigmoid and tanh as they can all approximate the same space of functions. The activation function for the output layer depends on the application. For our sentiment analysis task, we have multiple class labels which makes the softmax function useful as the activation function, defined as

$$\text{softmax}(\mathbf{l_j}) = \frac{\exp(\mathbf{l_j})}{\sum_{k=1}^{K} \exp(\mathrm{l_k})}, \tag{4.8}$$

where K is the number of output units, corresponding to the number of classes. When vectors are passed through the softmax function, their values are normalized to be between 0 and 1 and collectively sum up to 1, which means the output values can then be interpreted as probabilities:

$$p(c_k|\mathbf{x}) = \text{softmax}(\mathbf{l_k}). \tag{4.9}$$

After a forward pass during training, the accuracy of the output prediction is evaluated against the ground-truth class label. This is done using a loss function, also called an error function. A typical loss function is the cross entropy function, defined as

$$E = \frac{1}{N} \sum_{n=1}^{N} [y_n \log q_n + (1 - y_n) \log(1 - q_n)], \tag{4.10}$$

where y_n is the ground-truth class label for input $\mathbf{x_n}$ and q_n is the predicted class label. This loss function indicates the accuracy of the predictions, or rather the degree of the error. To get the best possible predictions, we minimize this function by optimizing the weights in the network. This is done by propagating backwards through the network, by calculating the derivative of the loss function with respect to each of the weights

$$\frac{\partial E}{\partial w_{jj'}} = \delta_j l_{j'} \tag{4.11}$$

and then updating the weights using gradient descent. To efficiently calculate these derivatives and get the gradients δ_j, a technique known as backpropagation is used [32]. The gradients outputted by the backpropagation algorithm denotes the slope of the function, which indicates in which direction the weights must be updated, and updating using gradient descent means updating in the descending direction of the slope, i.e. the direction in which the error decreases the most [31]. This is done from the end of the network, calculating the derivatives of the loss function with respect to each output unit and then the gradients are propagated backwards through the network, repeating these calculations with the respect to the weights of each layer:

$$\delta_j = \frac{\partial E}{\partial l_j} \tag{4.12}$$

4.3 Embeddings

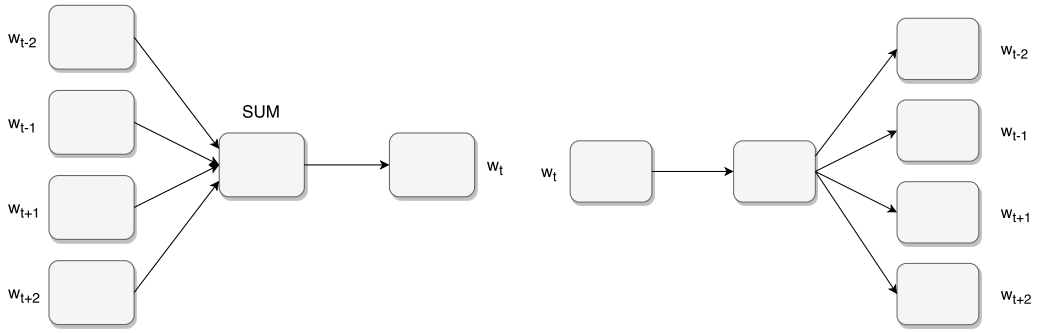
To classify the sentiment of the forum posts, we need to extract features from the text and represent them such that they can be processed mathematically. The general approach for this, is to encode the text as a fixed-length feature vector. A simple fixed-length feature could be a bag-of words model, where the text is represented as the multiset of its words, where the multiplicity, or word frequency, is a key feature. However, this approach has the disadvantage of discarding the grammatical and semantic properties of the text, even though such properties are useful for sentiment analysis or general text classification [33].

A model that aims to solve these shortcomings, is *Word Vector*, proposed by Mikolov et al.[33, 5]. This is an unsupervised algorithm that learns fixed-length feature representations of words, by training a neural network to predict word occurrence in the text. The core idea of Word Vector is to turn a corpus of raw text into a supervised learning task, by using words and their context as the features. The task then becomes predicting the correct word in a context from words sampled from the corpus.

Mikolov et al. has proposed two different models for this task, the Continuous Bag-Of-Words (CBOW) and the Skip-gram model model. The two models are algorithmically similar, but works inversely in regards to input and output. CBOW predicts a target word from a set of context words, while the skip-gram model predicts context words from a target word. An illustration of this concept can be seen in Figure 4.2. More formally at the input layer for both models, words are inputted using a 1-of-V encoding (one-hot encoding), where V is size of the vocabulary. The CBOW model takes multiple of these vectors as inputs, which are then projected into the same position in a shared projection layer, averaging the vectors of the input words. The input words are the words surrounding a

target word and the training criterion is to correctly classify the target word in the middle. For Skip-gram, the target word is taken as an input and projected into a continuous projection layer. The output layer is then replicated multiple times and a prediction is made for each context words.

This has the effect that, because CBOW takes a group of words as input, it treats an entire context as one observation and smoothes over a lot of distributional information. Skip-gram on the other hand takes individual words as input, meaning each context-word pair is treated as a new observation. This works better than CBOW for large datasets, because less information is lost, while for smaller datasets it is advantageous using the more generalized approach of CBOW [5, 34].



(a) The Continuous Bag-Of-Words model. (b) The Skip-gram model.

Figure 4.2: Word vector models.

For representing documents, Le and Mikolov has also proposed an extension of this model called *Paragraph Vector* [12]. This method learns fixed-length feature representations from variable-length pieces of texts, such as sentences, paragraphs, and documents, through small extensions and restrictions to the word vector models.

For our target dependent sentiment analysis task, we wish to model both the words in a post and the posts in a thread. As a result we will use both Word Vector and Paragraph Vector to create embeddings of our dataset. Because we have a large dataset, we choose to use skip-gram as our word vector model. The Paragraph Vector extension of this is known as Distributed Bag-Of-Words, and we will use this for our documents vectors to keep the architectures behind training each type of vector consistent. We now describe the chosen paragraph vector model in detail, which also covers the underlying word vector model.

4.3.1 Distributed Bag-Of-Words

Le and Mikolov proposed two paragraph vector models: distributed memory (PV-DM) and distributed bag-of-words (PV-DBOW). PV-DBOW takes inspiration from

the Skip-gram model of Word Vector, where the task is to predict context words from a target word. The intuition is that, given a document as input, we want to sample a random text window from this document and predict the words of the window. This is illustrated in Figure **??**.

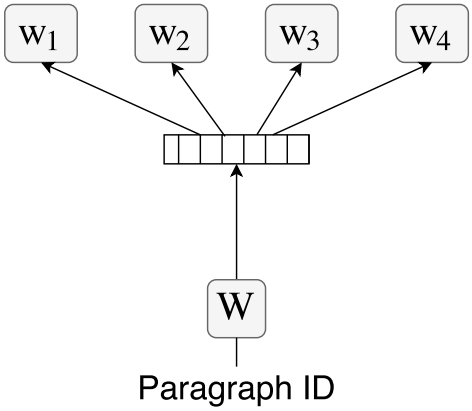


Figure 4.3: The Distributed Bag of Words version of paragraph vector[12].

The size of the input is based on the number of documents in the corpus. This means the layer has D units, where D is the number of documents. Together these units form a document vector $\mathbf{d}$ which is propagated forward to the next layer when training. This vector is represented using a one-hot encoding where each unit corresponds to a document id. For a specific document, this means that the unit corresponding to the document id is set to 1, and all other units are set to 0.

The size of the hidden layer, N, determines the dimensionality of the paragraph vectors. This is a hyperparameter that can be changed depending on the application. This activation is simply a weighted sum of the activation in the input layer, defined as

$$\mathbf{h} = \mathbf{d}^\top \mathbf{W}, \tag{4.13}$$

where $\mathbf{W}$ is the $D \times N$ weight matrix and $\mathbf{h}$ is the paragraph vector produced in the hidden layer. Because the input vector is one-hot encoded this corresponds to the hidden vector becoming a single row from the weight matrix. For instance, if $\mathbf{d} = [0, 1, 0, 0]$ and $\mathbf{W}$ is a 4×3 matrix with weights

$$\mathbf{W} = \begin{pmatrix} 0.013262 & -0.782341 & 0.563425 \\ 0.894377 & 0.674544 & 0.028133 \\ -0.556324 & 0.092939 & -0.348780 \\ 0.447887 & -0.093245 & 0.723945 \end{pmatrix}, \tag{4.14}$$

the hidden vector is computed as:

$$\mathbf{h} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \times \begin{pmatrix} 0.013262 & -0.782341 & 0.563425 \\ 0.894377 & 0.674544 & 0.028133 \\ -0.556324 & 0.092939 & -0.348780 \\ 0.447887 & -0.093245 & 0.723945 \end{pmatrix} = \begin{pmatrix} 0.894377 & 0.674544 & 0.028133 \end{pmatrix} \tag{4.15}$$

Finally the size of the output layer is defined by number of unique words in the vocabulary across all documents in the corpus. Because we predict multiple words at each iteration the output layer is replicated multiple times, one for each word in the window. Thus, each output layer has M units, where M is the number of words. As with the connection between the input and hidden layer, the hidden and output layers are fully connected and each connection has a weight. The output vector thus becomes

$$\mathbf{o} = \mathbf{h}^\top \mathbf{V}, \tag{4.16}$$

where $\mathbf{V}$ is the $N \times M$ weight matrix. A full illustration of this model can be seen in Figure 4.4

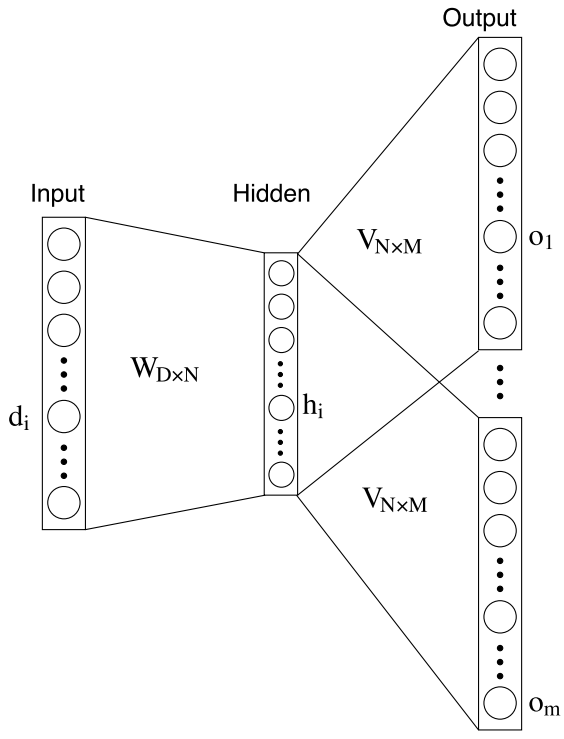


Figure 4.4: Neural network structure of the distributed bag-of-words model. The lines between the corners of the layers denotes that the layers are fully connected.

To make the predictions, we pass the output values, i.e. the i'th component of the the output vector $\mathbf{o}$, through a softmax layer. Applying the softmax function defined in Section 4.2 we get:

$$\mathbf{q_i} = softmax(o_i) = \frac{\exp(\mathrm{o_i})}{\sum_{m=1}^{M} \exp(\mathrm{o_m})}, \tag{4.17}$$

which means that the output vector now resembles an estimated probability distribution over the words of a document. Computing the full softmax very computationally expensive, as M is the size of the word vocabulary, which means a very large amount of weights has to be updated after each prediction. To optimize this, Mikolov et al. introduces negative sampling, which makes each training sample only modify a small percentage of the weights, rather than all of them, by introducing a few number of noise words (negative samples) to compare with.

To train the network to make good predictions, it must be measured how well the probabilities were estimated in regards to the true words of the document. This is done by comparing the prediction vector $\mathbf{q}$ with the target word vector $\mathbf{t}$. A single target word is predicted at a time, thus $\mathbf{t}$ is another one-hot encoded vector. A loss function is used to compare $\mathbf{q}$ and $\mathbf{t}$, to measure how accurate the prediction was. The loss function is a variant of cross-entropy defined as:

$$E(\mathbf{q}, \mathbf{t}) = -\sum_i t_i log(q_i), \tag{4.18}$$

During training the model is trained with a set of training examples. To calculate the overall loss, the cross-entropy loss over the training data is averaged:

$$Loss = \frac{1}{|T|} \sum_{j=i}^{|T|} E(\mathbf{q}, \mathbf{t}), \tag{4.19}$$

where T is the set of target vectors and $\mathbf{t_j}$ is the j'th target vector. This is the value we want to minimize.

The amount of loss resulting from the loss function is reflected by how the weights are configured in the two weight matrices W and V. These weights are typically initialized to random low values as a starting point. The goal then becomes to find the configuration of weights that yields the lowest possible loss. Like a standard neural network, this optimization process is done using a gradient descent based method and backpropagation.

Training Data

In addition to negative sampling, Mikolov et al. has proposed a couple of other optimization methods, to both increase efficiency and produce better vectors. This is done with the assumption that some words are less relevant than others and

can thus be removed, reducing vocabulary size. Some words are very infrequent and will be very hard to predict correctly as they have little or no co-occurrences. These words can be handled by simply discarding them, or by mapping them to placeholder word if it makes sense to preserve their location in the word window. Likewise, there are words that are so common, that they provide little semantic value. Similarly to how a TF-IDF model normalizes term frequencies with an overall document frequency for the term, the skip-gram model subsamples high-frequency words and assign lower probabilities to them[5].

Each word w_i in the training set is discarded with probability computed by the following equation:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{4.20}$$

where $f(wi)$ is the frequency of word wi and t is a chosen threshold.

4.4 Recurrent Neural Networks

So far we have introduced the standard feedforward neural network and shown their power for language modelling. However, they do have some limitations. A standard feedforward neural network is modelled with the assumption that each input example is independent of the others, which means that whenever an example has been processed, the state of the network is reset. This is not a problem if the examples are truly independent, but it means that sequential data such as sentences, which we work with, cannot be modelled by this type of network. For instance, if you want to predict the next word in the sentence "*In Denmark people speak...*", you know the answer because you read the previous words in the sentence. Recurrent Neural Networks (RNNs) fix this problem by introducing a notion of time steps, allowing for selectively passing information across sequence steps, processing the sequential data one step at a time [30].

RNNs is a variation of feedforward networks, extended with edges connecting adjacent time steps. This can be seen as a simple network with one input layer, one output layer, and recurrent layer, as illustrated in Figure 4.5. The loop in the recurrent layer across time steps can be visualized by unrolling the network. It can simply be seen as copies of the same small network, that passes information to its successor, as illustrated in Figure 4.6.

RNNs have a lot of similarities with standard feedforward Neural Networks (NNs), as seen by Figure 4.6. The most notable difference is that RNNs have two different inputs at each time step. One input directly from the input layer and an additional input from the hidden layer activations from the previous time step. Each of the connections also has weights, which are shared across all the time steps. The three shared weight matrices $\mathbf{U}$, $\mathbf{W}$, and $\mathbf{V}$ are shared for the input

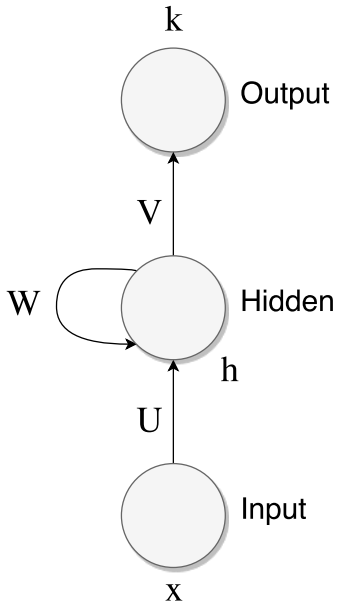


Figure 4.5: A recurrent neural network.

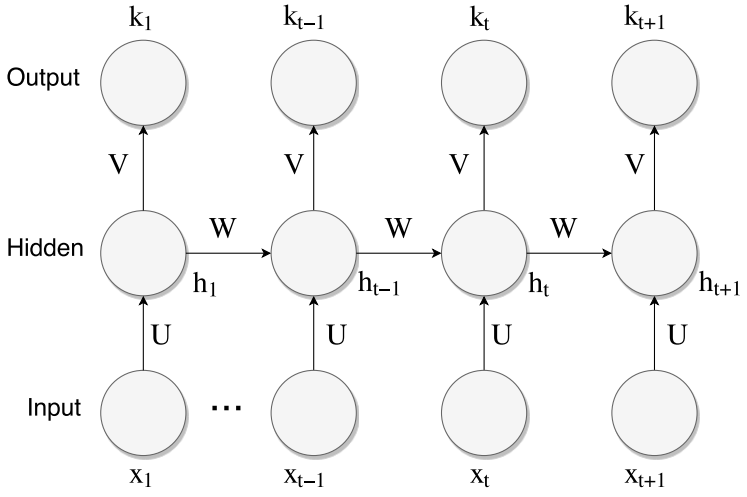


Figure 4.6: An unfolded visualization of a recurrent neural network. Each node represents a layer of network units at a single timestep.

layers, hidden layers across time steps, and output layers respectively. The input of a RNN is a sequence of length T. We define $\mathbf{x_t}$ as the input vector i at time t, $\mathbf{h_t}$ as the hidden vector h at time t, and finally $\mathbf{k_t}$ as the output vector k at time t.

Recall that in the feedforward network a weighted sum of the inputs is calculated and then an activation function is applied. Exactly the same is done for the RNN, with the only difference being, that a weighted sum is calculated for both types of inputs and thereafter they are added together:

$$\mathbf{h_t} = a(\mathbf{U}\mathbf{x_t} + \mathbf{W}\mathbf{h_{t-1}}). \tag{4.21}$$

This is performed recursively from time step $t = 1$ to $t = T$, to process the entire sequence. At each step the output is also calculated by a weighted sum of $\mathbf{h_t}$

$$\mathbf{k_t} = a(\mathbf{V}\mathbf{h_t}). \tag{4.22}$$

For the output activations, the same activation and loss functions can be used, as in a standard feedforward NN, described in Section 4.2.

The prediction we wish to consider can vary depending on the application, as the network has an output layer at each time step. In the context of target dependent sentiment analysis there are multiple ways this can be handled. The output at each time step represents the output for the sequence of words processed so far. If we consider the target to be the end of the sequence, we will only be interested in the final output of the network. On the other hand if we consider the target word to be mid-sequence, we are also interested in the output for that particular time step.

As with the standard feedforward NN we also want to optimize the weights of the network by calculating the derivative of the loss function with respect to each of the weights. Because of the added element of time in RNN, an extension to backpropagation called *Backpropagation Through Time* (BPTT) [35] is used. It functions exactly the same way as standard backpropagation, by going through the whole network in reverse, doing a repeated application of the chain rule. There is simply an extension of the series of functions for which derivatives are calculated.

The most important thing to note is that for RNNs the loss function depends on the activation of the hidden layer not only through its influence on the output layer, but also through its influence on the hidden layer at the next timestep:

$$\frac{\partial E_T}{\partial \mathbf{W}} = \frac{\partial E_T}{\partial \mathbf{k_T}} \frac{\partial \mathbf{k_T}}{\partial h_T} \frac{\partial \mathbf{h_T}}{\partial \mathbf{W}} \tag{4.23}$$

where $\mathbf{h_T} = a(\mathbf{U}\mathbf{x_T} + \mathbf{W}\mathbf{h_2})$ depends on $\mathbf{h_t} = a(\mathbf{U}\mathbf{x_t} + \mathbf{W}\mathbf{h_1})$, which again depends on $\mathbf{W}$ and $\mathbf{h_{t-1}}$ and so on. With this chain in mind the previous equation becomes:

$$\frac{\partial E_T}{\partial \mathbf{W}} = \sum_{j=1}^{T} \frac{\partial E_T}{\partial \mathbf{k_T}} \frac{\partial \mathbf{k_T}}{\partial \mathbf{h_T}} \frac{\partial \mathbf{h_T}}{\partial \mathbf{h_j}} \frac{\partial \mathbf{h_j}}{\partial \mathbf{W}}. \tag{4.24}$$

This means that to propagate backwards and compute the whole sequence of gradients, we start at $t = T$ and recursively calculate the gradients for the hidden layer across time steps, decrementing t at each step. This is illustrated in Figure 4.7.

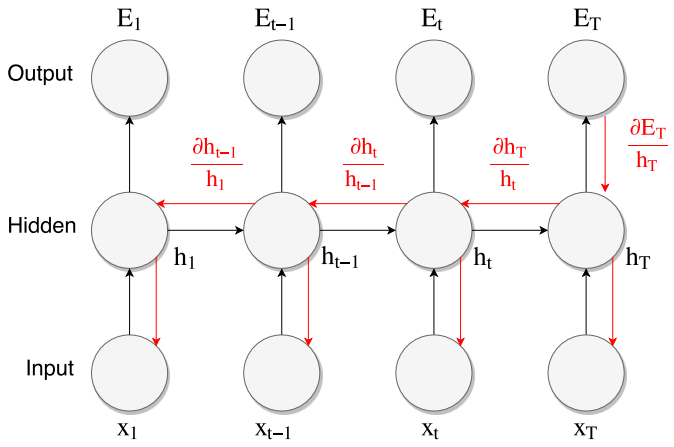


Figure 4.7: BPTT in an unfolded RNN.

Since the weights of **W** are shared for each step the contributions of each time step to the gradient are simply summed. The input weights **U** has similar dependencies, while calculating the gradient of **V** follows standard backpropagation.

4.4.1 The Vanishing and Exploding Gradient Problems

A problem with standard RNNs, is that the range of the sequences they can model has been shown to be very limited, i.e. they have a hard time learning long-term depedencies. The issue arises when the influence of a given input on the hidden layer and the output passes back through the recurrent connections. The associated gradient then have a tendency to either decay or explode exponentially. This is known as the vanishing gradient and exploding gradient problem, respectively [36, 37]. The problem is that the sensitivity to a given input decays over time as new inputs overwrite the activations of the hidden layer and the weights have decreasing influence on the output. This means that during backpropagation, when we look at the first input at the end of the sequence, this information will be forgotten after multiple other inputs have been processed. This is illustrated in Figure 4.8.

With exploding gradients the problem is the opposite and happens when gradients get exponentially large when being multiplied by numbers bigger than one.

In (4.25) the gradient at each step $\frac{\partial \mathbf{h_T}}{\partial \mathbf{h_j}}$, can each be derived using the chain rule, for example:

$$\frac{\partial \mathbf{h_T}}{\partial \mathbf{h_{t-1}}} = \frac{\partial \mathbf{h_T}}{\partial \mathbf{h_t}} \frac{\partial \mathbf{h_t}}{\partial \mathbf{h_{t-1}}}. \tag{4.25}$$

This can be rewritten into a sum-of-products form:

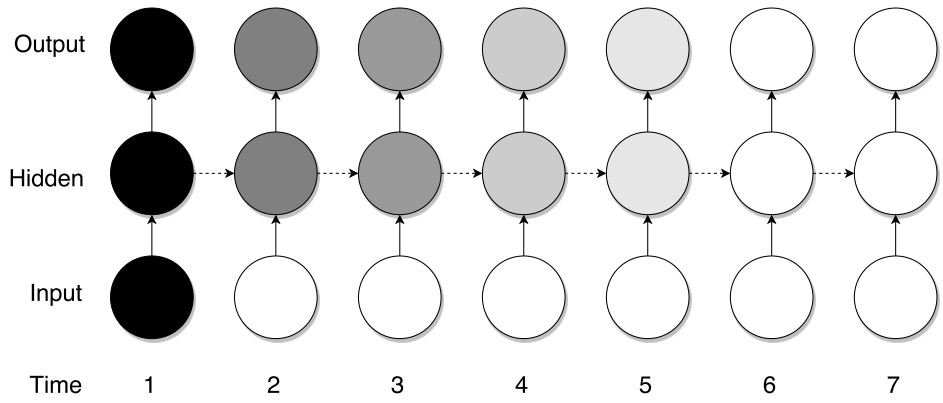


Figure 4.8: The vanished gradient problem. The shading indicates the nodes' sensitivity to the initial input, where darker colours means more sensitivity. The sensitivity then decays and gets weaker over the time steps.

$$\frac{\partial E_T}{\partial \mathbf{W}} = \sum_{j=1}^{T} \frac{\partial E_T}{\partial \mathbf{k_T}} \frac{\partial \mathbf{k_T}}{\partial \mathbf{h_T}} \Big(\prod_{g=j+1}^{T} \frac{\partial \mathbf{h_g}}{\partial \mathbf{h_{g-1}}} \Big) \frac{\partial \mathbf{h_j}}{\partial \mathbf{W}}. \tag{4.26}$$

Each of these results are matrices. What happens when there are many times steps and the weights are either very small or very large, is that they will exponentially shrink towards 0 or explode towards infinity across matrix multiplications. Pascanu, Mikolov, and Bengio proves that this can happen both when the largest singular value λ_1 of $\mathbf{W}$ is larger than $\frac{1}{\gamma}$, where γ is the bound value of the derivative a'.

Numerous attempts has been made to solve the problem of vanished and exploding gradients for RNN. One solution to exploding gradients is to normalize the gradients

$$\delta = \frac{threshold}{\|\delta\|}\delta \tag{4.27}$$

when their norm exceeds a certain threshold, *threshold*. This is known as gradient clipping [38]. A popular solution to vanishing gradients is ReLU, because it does not bound the output, thus keeping the gradient constant and not increasingly small or large over time steps. This comes from the function being piece-wise linear. If $x \leq 0$ then $f(x) = 0$ for which the derivative is $f'(x) = 0$. If $x > 0$ then $f(x) = x$ for which the derivative is $f'(x) = 1$. Another popular solution to both problems is the Long Short-Term Memory architecture.

4.4.2 Long Short-Term Memory

Long Short-Term Memory (LSTM) networks was introduced by Hochreiter and Schmidhuber in [11]. This architecture is explicitly designed to avoid the long-

term dependency problem of standard RNNs. Thus, LSTMs is very useful for our domain where posts are sometimes very long. It is similar to a standard RNNs with one hidden layer. The main difference however, is that the nodes in the hidden layer are replaced with so-called memory blocks, which themselves consists of multiple components. Each memory block has a self-connected recurrent edge of a fixed weight, ensuring that the gradient can pass across many time steps without vanishing or exploding. The memory blocks are made up of a cell state and multiple gates. The cell state keeps track of information and the gates controls the flow of the information, by controlling what information to keep or discard. The gates are called as such because they are sigmoidal functions which compresses their input to between 0 and 1, and by pointwise multiplication with another vector they define how much of that vector is kept. This replaces the shared weights of a standard RNN with different weights for each gate, and a constant weight for the cell state update [30]. A visualisation of an LSTM block can be seen in Figure 4.9.

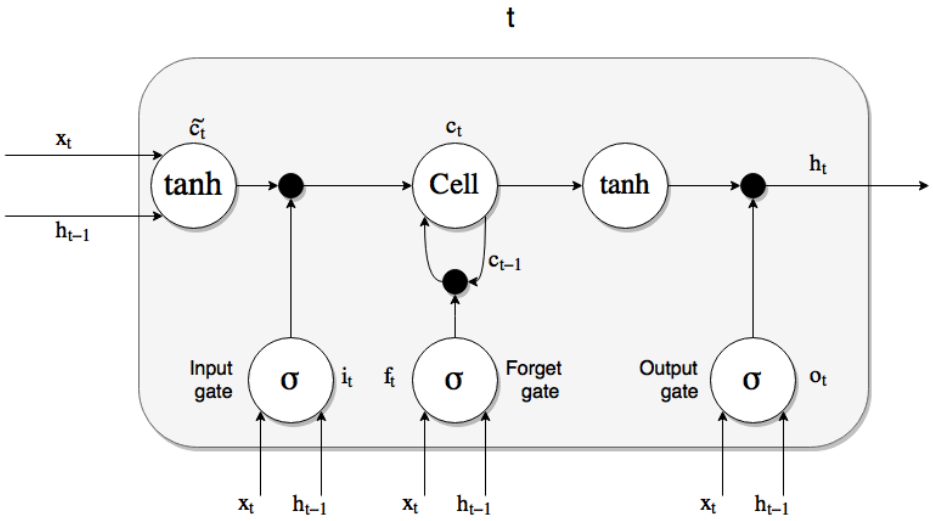


Figure 4.9: A single LSTM block. The input, forget and output gates are non-linear summation units that collect activations from inside and outside the block, and control the activation of the cell through multiplications, denoted as black circles.

The activations of the different gates depends both on the current input layer and the output of the previous memory block, similar as the hidden layer of a standard RNN. This means an LSTM can be visualized as an unrolled RNN, as seen in Figure 4.10.

As with the previously described networks, LSTMs also performs a forward and backward pass to optimize its weights. At each time step, we start by deciding what previous information to discard. This is done by passing the previous hidden output representation $\mathbf{h}_{t-1}$ and the current input $\mathbf{x}_t$ through the forget gate, which

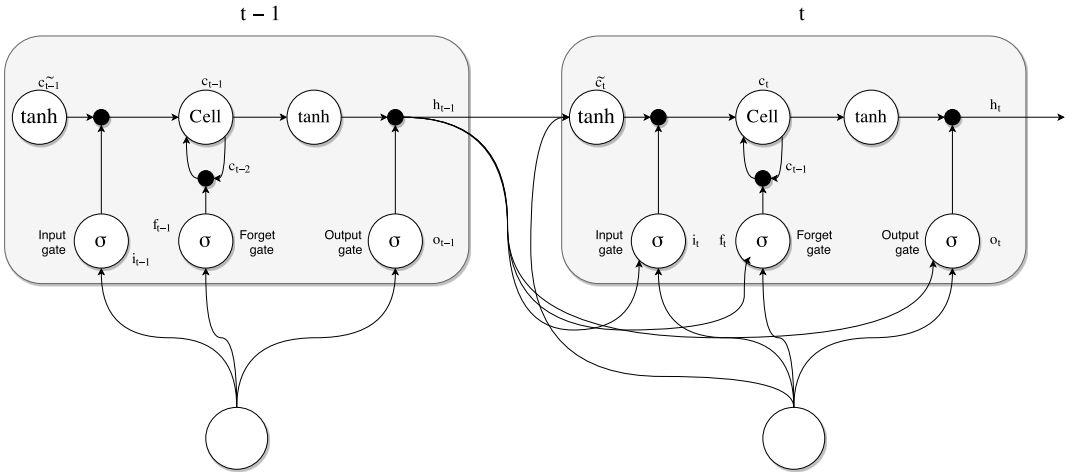


Figure 4.10: Multiple connected LSTM blocks. Information is passed through successive blocks, as input to the different gates along with the input at each time step.

outputs a value between 0 and 1 for each component of the previous cell state vector $\mathbf{c_{t-1}}$, where 1 preservers all information and 0 preserves none. The resulting function is defined as [30]

$$\mathbf{f_t} = \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]), \tag{4.28}$$

where $\mathbf{W}_f$ is the weighting of the forget gate. This step can be seen in Figure 4.9 at the forget gate. Next, we must decide which information to add to the cell state. This is done by first deciding which values to update using another sigmoid layer called the input gate, and then compute a candidate vector $\tilde{\mathbf{c}}$ of potential values using a hyperbolic tangent layer, see Figure 4.9, "Input gate" and $\tilde{\mathbf{c}}_\mathbf{t}$,

$$\mathbf{i_t} = \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]) \tag{4.29}$$

$$\tilde{\mathbf{c}}_\mathbf{t} = tanh(\mathbf{W}_c \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]). \tag{4.30}$$

Updating the cell state from $\mathbf{c_{t-1}}$ to $\mathbf{c_t}$ is then done by combining these functions. The pointwise product of the old state $\mathbf{c_{t-1}}$ and $\mathbf{f_t}$, is computed to discard information, and then we add the pointwise product of $\mathbf{i_t}$ and $\tilde{\mathbf{c}}_\mathbf{t}$, which are the candidate values, scaled by the same sigmoid function used in the forget gate:

$$\mathbf{c_t} = \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \tilde{\mathbf{c}}_\mathbf{t}. \tag{4.31}$$

This step can be seen in Figure 4.9 as the incoming edges to the cell block, where the black circles denotes the pointwise product, ($\odot$ in the equation). In the same manner as the forget and input gate, a sigmoid layer is used to decide which parts

of the cell state to output, and a hyperbolic tangent layer is then used to normalize the output values. The outputted hidden representation $\mathbf{h}_t$ of the cell thus becomes:

$$\mathbf{o_t} = \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]) \tag{4.32}$$

$$\mathbf{h}_t = \mathbf{o_t} \odot tanh(\mathbf{c_t}), \tag{4.33}$$

as illustrated by the rightmost part of Figure 4.9. As all the gates have the same equations, only with different weight matrices, the equations can be re-parametrised to

$$\begin{pmatrix} \mathbf{f} \\ \mathbf{i} \\ \tilde{\mathbf{c}} \\ \mathbf{o} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ tanh \\ \sigma \end{pmatrix} (\begin{pmatrix} \mathbf{x_t} \\ \mathbf{h_{t-1}} \end{pmatrix} \cdot \mathbf{W}) \tag{4.34}$$

Like a standard RNN this represents the sequence so far at each time step. For our objective, where the target is the end of the sequence, the final outputted vector $\mathbf{h_T}$ is the output we are interested in.

The backward pass follows the same principle as a standard recurrent neural network. It is calculated using BPTT, starting at the end of the input sequence, recursively calculating the derivatives while progressing back through the sequence. However, it is important to notice the order of the backpropagation through the different components inside the LSTM block. Firstly, the gradient of outputted hidden representation of the cell, which is passed back to the output gate and combined with the differentiated sigmoid function in the layer. Secondly, the current cell state receives both the output gradient and the gradients of the following cell state. These two are then accumulated and backpropagated through the previous cells [39]. Finally, the derivatives are computed for the forget gate and input gate, before moving on the next time step.

4.5 Our Models

We propose three different models, which uses LSTM networks to process the contents of the forum threads sequentially, using the embedding methods presented in Section 4.3 [5, 12]. The initial model design is inspired by the Target Dependent LSTM model by Tang et al. in [4]. As described in Section 2, they utilize LSTMs to model the preceding and following contexts surrounding the target phrase. Thus, the textural context in both directions can be used as feature representations which takes the target into account.

We wish to expand this idea of combining two different directional LSTMs, by incorporating the natural structure of forum threads, where threads are made up of posts and posts are made up of words. Thus, in addition to processing the context

words of the target, we also process the context posts surrounding the target post. The intuition is that the local context is strongly connected to the target, but also modelling the global discussion can serve as a booster or normalizer, and with these extra features, the combined sentiment .

As mentioned we use word and document embeddings to represent our text as fixed-length feature vectors. We can incorporate these into our model either as learned vectors or pre-trained vectors. Learned vectors means that the embeddings are learned as part of our model, at the same time as the sentiment prediction, and are optimized using the same loss function. This has the advantage that the resulting word embeddings are optimized and grouped in regards to the sentiment classification task. Pre-trained vectors on the other hand are word embeddings either trained on a different dataset, or trained on our data separately, before the sentiment classification model is trained. This is advantageously as the general vector representation on our data, can be reused for multiple purposes or models. Additionally, the vectors remains consistent every time we perform sentiment classification. We choose to split the task of learning the embeddings and the sentiment classification, Thus, we pre-train the embeddings on our data and afterwards use them as input to our model. We chose this due to the described benefits, and to limit the number of parameters of our models, resulting in faster training.

Below we describe our two proposed models, Flow-LSTM and Split-LSTM, which models the hierarchical thread structure with the purpose of target dependent sentiment classification in two different ways.

4.5.1 Flow-LSTM

Flow-LSTM directly expands on the idea of the Target Dependent LSTM model by Tang et al. The basic idea is to model the preceding and following contexts surrounding the target phrase, so that contexts in both directions are used as feature representations for sentiment classification. The context in this case consists both of the posts surrounding the post containing the target words, as well as the words inside that post. For this model we simply view the global context of the surrounding posts as an extension of the local context words. Hence, we process the document embeddings of the posts as part of the the same sequence as the word embeddings of the target post. In other word, the input sequence to the model start with the document embeddings and transitions directly into the word embeddings. This concept is illustrated in Figure 4.11.

The overall input of the deep neural network is a thread consisting of m context posts, $\{p_1, p_2, \ldots, p_m\}$, and a target post $p_1 \leq p_t \leq p_m$ consisting of n words, $\{w_1, w_2 \ldots, w_n\}$. Inside the target post is a target phrase $\{w_{l+1}, w_{l+2}, \ldots, w_{r-1}\}$, a sequence of preceding context words $\{w_1, w_2, \ldots, w_l\}$ and a sequence of following context words $\{w_r, \ldots, w_{n-1}, w_n\}$. Thus, a thread is split into five components: preceding posts, preceding context words, target words, following context words,

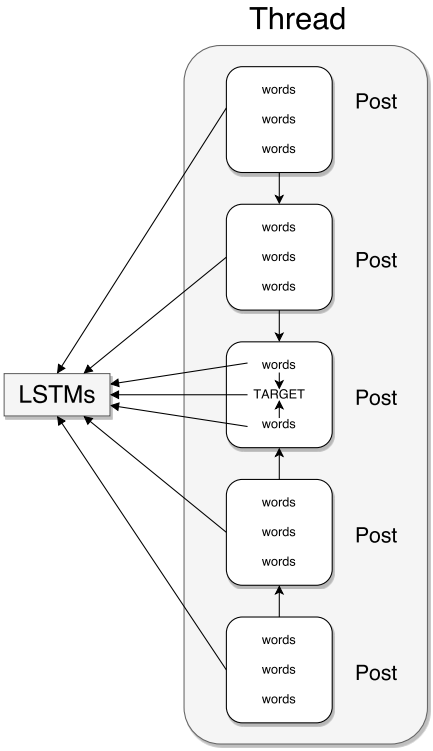


Figure 4.11: Overview of the concept of Flow-LSTM.

following posts. The LSTM progresses sequentially through the context, where each LSTM block takes a new embedding and the previous context representation as input, and outputs a new hidden vector representing the thread so far. This is done using LSTMs in both directions towards the target words. An LSTM_L process the preceding posts concatenated with the preceding words, and an LSTM_R does the same in reverse order for the following posts and words. When we have the final hidden vectors, representing the entire context, we do a linear combination, which can either be concatenating, summing or averaging the vectors, denoted $\oplus$

$$\mathbf{h}_{r-1}^{L} \oplus \mathbf{h}_{l+1}^{R}, \tag{4.35}$$

and feed them to a linear layer. The length of the output of this linear layer, is the number of polarity classes, and feeding this output to a softmax layer outputs the probability of classifying the polarity of the target as positive, negative or neutral. This is illustrated in Figure 4.12.

We train this model in an end-to-end way and optimize it in regards to the target dependent labels as our ground-truth polarity classes. Training the network is done exactly like a standard LSTM, with the only addition being the linear combination of the two LSTMs. Given the gradient of the output with respect to

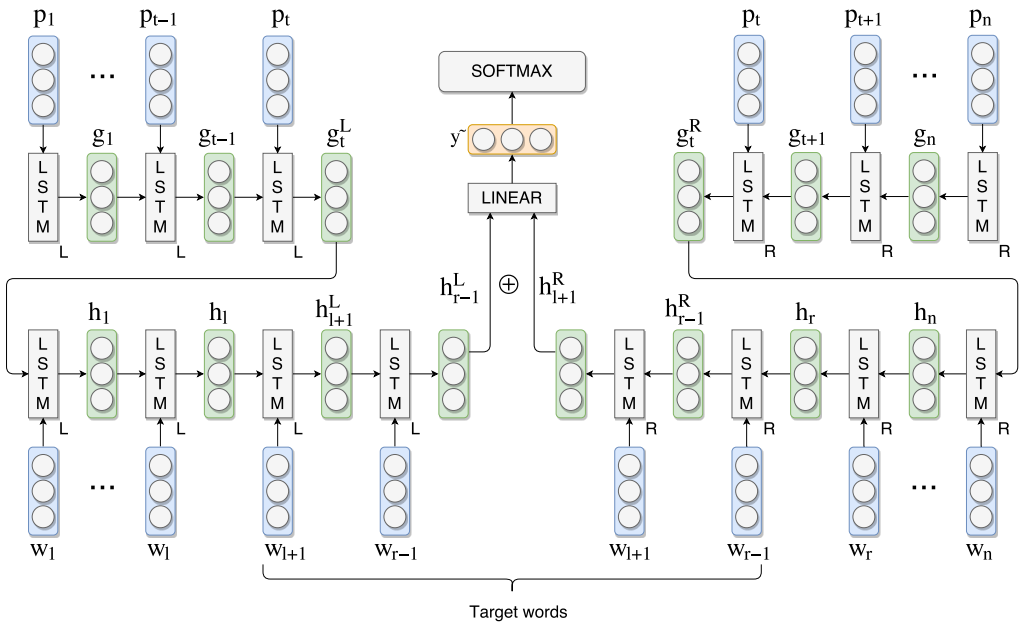


Figure 4.12: The network architecture of Flow-LSTM. The inputs are word vectors **w** and document vectors **p**. **h** and **g** represents hidden vectors.

our loss function, we compute the gradient of the input to the softmax layer. This is then accumulated with the output gradients of the respective LSTMs which update their weights individually.

Flow-LSTM is a simple way to extend the local context words with the global context posts. It is based on the assumption, that since both classes of embeddings are trained using the same algorithm, that they are compatible as inputs to the same network, and of course, that considering a wider context can improve the performance compared to only considering the target post itself. We keep the dimensions of both types of embeddings at the same size and train them on the same word vocabulary.

4.5.2 Split-LSTM

Split-LSTM aims to model the natural hierarchy in a thread as separate entities and then combine them into a target-dependent prediction. It is based on the assumption that the local context within the target post serves as a strong indicator of the target sentiment, while the global context of the thread expresses the general sentiment of the thread, such that the global content can be used to boost or normalize the target sentiment score.

Split-LSTM is similar to Flow-LSTM, where preceding and following contexts surrounding the target string are used for making a target-dependent sentiment

prediction. The difference is, that while Flow-LSTM considers the global context to be an extension of the local context, Split-LSTM considers them as two separate contexts. Thus, Split-LSTM processes posts and words separately in different sub-networks, and then combine them at the end using a weighted linear combination, see figure **??**. The result is a hierarchical structure of neural networks in which we

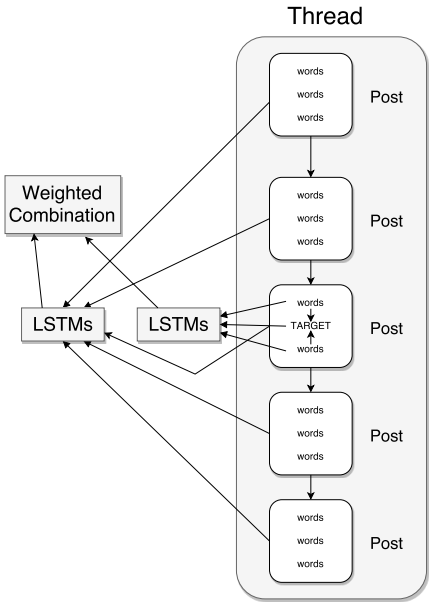


Figure 4.13: Overview of the concept of Split-LSTM. Word and post sequences are processed individually using different LSTM networks and are then combined using a weighted linear combination.

need to determine the optimal weights. There are two different ways to do this:

Individual: Optimize the weights in each LSTM individually, when making their respective predictions and then learn the weighting between them afterwards or determine it heuristically.

Combined: Optimize the weights throughout the entire network from the final prediction.

The second option means training the network in an end-to-end way, where both the parameters of the LSTMs and the parameters of their combination are learned. Figure 4.14 shows an illustration of this model. It also results in only utilizing the embeddings of the surrounding posts as additional context, but not their general labels. Thus, we only have one loss function regarding the target labels as our ground-truth polarity classes.

In the forward pass of this network the LSTM's process the contexts in different directions similar to Flow-LSTM. For the word-level sub-model, the preceding

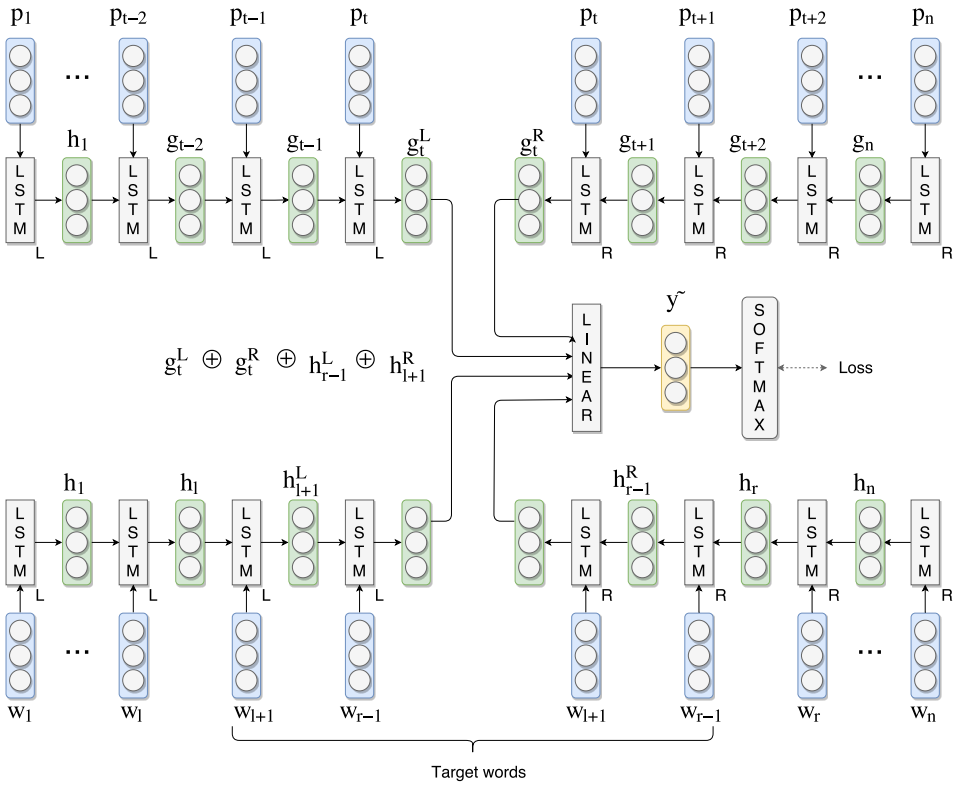


Figure 4.14: The architecture of Split-LSTM. It treats the different levels of the thread hierarchy as separate entities, rather than one long sequence.

words are processed towards the target using one LSTM and the following words using another in reverse direction. The same is done in the post-level sub-model, with the target post being the end of the sequences. Everything is then linearly combined,

$$\mathbf{g_t^L} \oplus \mathbf{g_t^R} \oplus \mathbf{h_{r-1}^L} \oplus \mathbf{h_{l+1}} \tag{4.36}$$

after which softmax is applied and a loss is computed with regard to target labels.

4.5.3 ML-LSTM-LSTM

ML-Split-LSTM, short for MultiLoss-Split-LSTM is an extension of Split-LSTM which aims combine both general and target dependent sentiment polarities into a final prediction, by evaluating both types of labels using multiple loss functions. Like Split-LSTM, it considers the word-level and post-level contexts to be separate entities which are then combined. The difference is at that instead of combining all four LSTM outputs at the same step, the word-level LSTM's and post-level LSTM's

are combined separately from each other. After each combination, a softmax layer is then applied from which a prediction is made and a loss is computed. For the word-level sub-model, the target dependent label is used for the prediction and for the post-level sub-model, we sum over the general labels for all the context posts and use the resulting label to compute another loss.

The resulting vectors from the softmax layers are then propagated forward and combined using a weighted linear combination. After this combination, we make a final prediction for the entire network using the target dependent labels. To sum up, we want to predict the target dependent label from the word-level context alone, the combined general label of the post-level context alone and finally predict the target dependent label, from the combination of the sub-models. This is illustrated in Figure 4.15.

Each of the loss functions return a loss regarding their respective labels. For the backpropagation in this network, we take a weighted sum of the resulting losses, and backpropagate through the network from the final layer, optimizing all the weight based on the summed loss function.

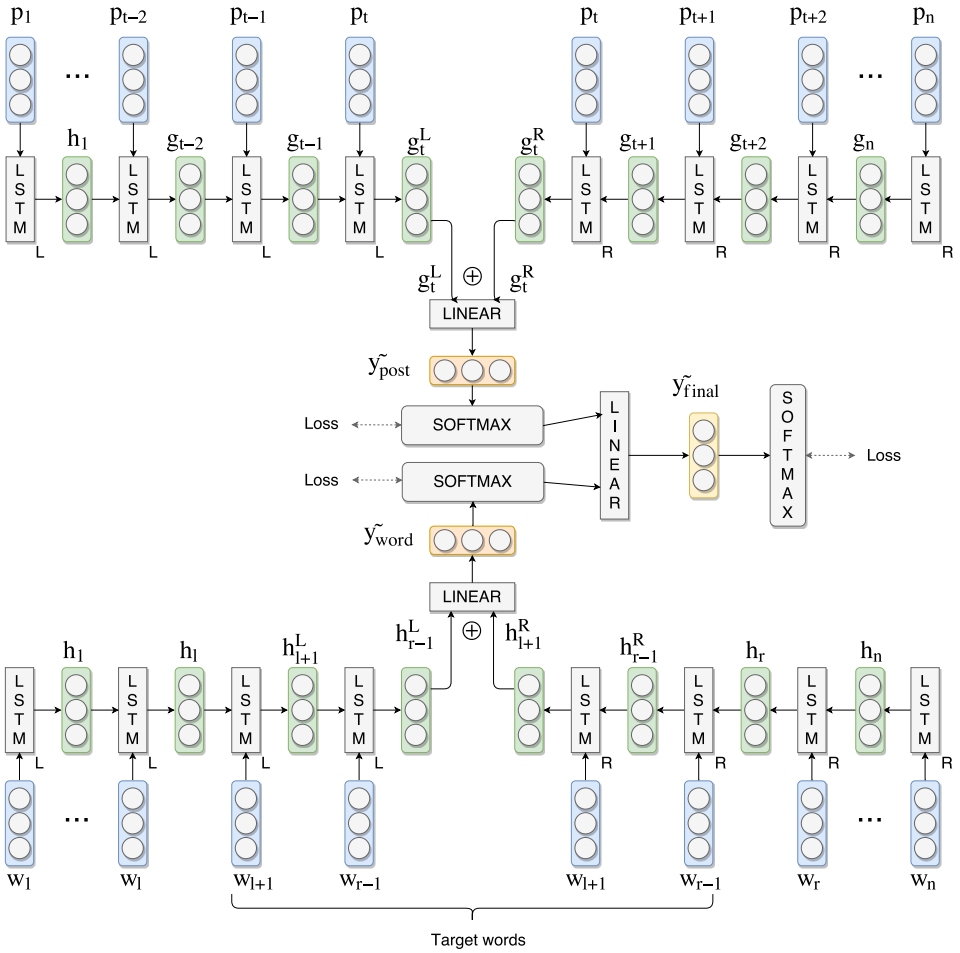


Figure 4.15: The architecture of ML-Split-LSTM. Like Split-LSTM, different levels of the thread hierarchy are processed separately and then combined into a final prediction. For ML-Split-LSTM losses are also computed at each submodel and the model is optimized from a weighted sum of all three loss functions.

Chapter 5

Experiments

We conduct multiple experiments with the purpose of evaluating the performance of our models on our manually labelled test sets. We compare our models to several different baselines, to examine whether taking targets into account is advantageous, and furthermore, if considering both local and global context of a post improves the accuracy compared to only the local context. We also evaluate how our models generalizes to supplements which have not been trained on, as well as the accuracy of classifying whether a target relates to cancer or not.

5.1 Experimental Data and Setting

The data used for evaluating our models is our manually annotated dataset. As described in Section 3.2.3, we split this data into multiple sets. For training the models we use a training set of 613 examples, each consisting of a target post and five context posts in each direction. This gives a label distribution of 12% negative, 34% neural, and 54% positive labels. As this dataset is unbalanced, we risk the models becoming biased towards two of the three classes. To test this hypothesis, we create a duplicate of the training set, where we oversample the negative labels, such that the distribution becomes 29% negative 27% neural, and 44% positive target labels. We run separate experiments on each of these datasets to evaluate how this oversampling affects performance.

To optimise the hyperparameters of our models, we perform 5-fold cross validation on the training sets, splitting the set into different combinations of 80% training data and 20% validation data at each fold. When we have determined the hyperparameter configurations for our models we train them on the full training set and evaluate their performance on a separate test set of size 150, with a class distribution of 19% negative, 28% neural, and 53% positive labels. Afterwards, we test how our model generalizes to un-trained supplements by evaluating the same model on another separate test set, containing only test examples for the new sup-

plement, calcium. This dataset has a size of 72 examples and a class distribution of 14% negative, 36% neutral, and 50% positive labels. For our evaluation metrics we use classification accuracy as well as macro-F1. Macro-F1 works as an extension from standard F-measure from binary to multiclass problems. For macro-averaging the F1-score, an F1-score is calculated for each individual class first and then averaged to get a single real number measurement [40]. Precision and Recall are thus calculated for each class, then the F-measure for each category k is computed and the macro-averaged F-measure is obtained by averaging F-measure values for each class:

$$F_k = \frac{2p_k r_k}{p_k + r_k}, \quad \text{Macro-F1} = \frac{\sum_{k=1}^{K} F_k}{K}, \tag{5.1}$$

where p_k and r_k are the precision and recall scores for class k. This metric is useful for problems with unbalanced classes, as it is highly influenced by the classifier's performance on rare categories, meaning that biased models will end up with low scores, even when having a high accuracy.

5.1.1 Model Input

As input to all of our models we utilize word and document embeddings, trained using Word2Vec for target posts and Doc2Vec for context posts. We pre-train these on our entire corpus, using the skip-gram model, see Section 4.3, and hyper parameter configurations based on the recommendations by Mikolov et al. For the context window they achieve the best results using a context window of 10 words for sentiment classification [12], which we do for our experiments as well. We chose a starting learning rate of 0.025 and decrease it linearly, so that it approaches zero at the end of the last training epoch, based on the results by Mikolov et al. We discard all words occurring less than 5 times in the corpus, to remove rare tokens such as spelling mistakes, and because including them will result in a very large model with many unique tokens. This results in retaining 351.376 unique tokens of the original 2.435.436 unique tokes. We also subsample frequent words using a threshold of 10^{-5}, as recommended by Mikolov et al., and described in Section 4.3, which further reduces the set of unique tokens by 2.374. For training we use negative sampling with 5 negative samples as recommended by Mikolov et al. for large datasets, also described in Section 4.3. Lastly, for the size we experiment with both 100, 200 and 300 dimensions for the vectors. We train the document embeddings on a corpus where the documents for the test set have been removed. For the test set we infer the vectors, by freezing the rest of the model, meaning the word vectors and the softmax weights, are fixed and then learning the document vector.

5.2 Models

In total we evaluate eight different models, consisting of our three proposed models and five different baseline models for comparison. Our baselines include:

Lexical analysis: A simple approach where the class label is derived from a lexicon of negative, neutral and positive words. Each word in a post is looked up with the lexicon, and the scores are then aggregated to get a total polarity for the post. For multiclass classification, this aggregation is done by returning the most occurring polarity of the post. This method does not consider the semantics and context of the text, and nor does it consider target words. This is included to evaluate the accuracy of the simplest possible baseline for this task. For this method an extensive list of words with associated polarities from the University of Pittsburg's subjectivity lexicon is used for scoring [41]. This lexicon includes 4154 negative, 447 neural, and 2304 positive words.

Handcrafted lexical analysis: An extension of the of lexical analysis methods, provided by Enversion A/S, intended to be a more domain-specific version of the previous method. Instead of using the full lexicon it uses a handpicked subset of the words. The handpicked lists were derived by retrieving the 500 most occurring words of each class, and choosing whether to keep or discard each one. This result in a list of 66 negative, 447 neutral, and 89 positive words. Note that this method was initially intended for binary classification of negative/positive and thus the neutral list has not been evaluated, and as a result the full list is used.

Multilayer Perceptron w/ Paragraph Vector: An established classification algorithm, using the document embedding vector of the target post as input. Target words are thus not considered by the model. We include this baseline method to evaluate the accuracy of a target-independent sentiment classification algorithm on the task of target dependent sentiment classification. To choose the classifier, we evaluated four different classifiers and choose the one with best accuracy on a validation set for our experiments. The classifiers evaluated were Logistic Regression, Naive Bayes, Multi-Layer Perceptron (MLP) and Support Vector Machines, where MLP achieved the best results, as shown in Appendix A.3.

LSTM: As our proposed models utilizes the LSTM architecture, we want to compare them to a standard LSTM model, which processes the words of the target post from start to finish in a target-independent way. We set a bound of 200 words for the input, based on the input length of our models, which uses 100 words on each side of the target. We run the LSTM with a hidden

layer equal to the size of the input embeddings and tune only the number of epochs for the hyperparameters.

Target-dependent LSTM (TD-LSTM): The model introduced by Tang et al., processes the target words in a post in both directions to perform target dependent sentiment classification [4]. We use this model as a baseline method for target-dependent sentiment analysis and to evaluate if including the global context of a target post, will improve the accuracy compared to this model, which only uses words. We run the model with the hyperparameters reported by Tang et al., to compare with the exact model they present in their paper. This means using using stochastic gradient descent with a fixed learning rate of 0.01 for optimisation and only tune the epochs for best performance on our dataset.

To consider the global context, we conduct experiments using our proposed models, Flow-LSTM, Split-LSTM and ML-Split-LSTM. The overall setup for these are as follows:

Flow-LSTM: For this model we use a global context window of five posts on each side of the target post. It then receives one set of input embeddings for each LSTM, the left-side context for the left LSTM, and the right-side context in reverse order for the right LSTM. This consists of a sequence of the document embeddings for the context posts followed by the word embeddings for the context words. This model makes predictions with regards to the target labels only.

Split-LSTM: Like Flow-LSTM five posts on each side of the target post is used for the global context window. These are also processed by two LSTM's, but separate from the context words, which are also processed by two LSTMs. Like Flow-LSTM, only target labels are used.

ML-Split-LSTM: This model is built up exactly as Split-LSTM, with two LSTM's for words and two LSTM's for context posts. The difference from Split-LSTM is that we use the general sentiment label for each context post and aggregate them into one general label for the target post. We then compute a loss for both the word and post sub-models in addition to the final loss and use a weighted sum of them for the optimisation.

The details of the models can be seen in Section 4.5. To optimise the performance of our models, we tune the number of epochs each model run, as well as introducing dropout into the networks to handle potential overfitting. We experiment with combinations of two different types of dropout. The standard dropout introduced by Srivastava et al. randomly disables units in the network[42]. We add this to

our input layer, which can be seen as removing random input embeddings from each sequence, artificially creating more varied training examples. This means that the models will not receive the exact same version of a training example across epochs. We also experiment with applying dropout within the LSTM cells. This is performed by multiplying the input vectors to each gate with random masks, which drops certain components of the vectors. When these vectors are multiplied by each gate's respective weight matrix, this corresponds to dropping random rows from the weight matrix at each step [43].

As mentioned, TD-LSTM is optimized using stochastic gradient descent with a fixed learning rate, to stay consistent with the authors' findings. For our models we optimise the learning using RMSprop, which is an adaptive learning rate method proposed by Tieleman and Hinton [44]. In short it keeps an exponentially decaying average of squared gradients, which the learning rate is divided by, resulting in faster training [45].

5.3 Results

The main experiment and evaluation metric analyses the accuracy of performing target-dependent sentiment analysis on our four chosen supplements. Throughout the section we present intermediate results and argue for our iterative improvement of the models using results tables, learning graphs and confusion matrices.

5.3.1 Initial Experiments

As mentioned previously we evaluate our models by doing 5-fold cross validation on our training sets. We conduct the experiments on both the original labelled training set and a training set of oversampled negative labels. To get a random label distribution in each fold, training examples are randomly shuffled. As the oversampled dataset contains duplicates of the examples with negative labels, there might be cases where a copy of an example is both in the training and the evaluation dataset. This means that the models will most likely overfit some negative examples and generally perform better on negative labels and overall. We take this into account when tuning our models, and experiment with dropout to introduce inequalities into these examples.

For the initial experiment, the baselines and our models are evaluated using their mean training and evaluation accuracy, as well as the F1-macro score over the five folds, as presented in Table 5.1. The LSTM-based models is executed with a range of maximum 80 to 120 epochs, and their best produced mean evaluation accuracy is reported. As TD-LSTM utilizes stochastic gradient descent with a fixed learning rate of 0.01, convergence is much slower compared to the other LSTM models using RMSprop for optimisation. Therefore, TD-LSTM is executed with

epoch range of 200 to 250.

Table 5.1: Cross validation results of the models for 200 dimension embeddings

Models	Original			Oversampled		
	Train	**Val**	**F1**	**Train**	**Val**	**F1**
Full Lexicon-based	41.44	41.43	34.90	38.17	38.18	26.92
Handcrafted Lexicon-based	38.17	38.18	26.92	31.54	31.53	22.50
Paragraph Vector (MLP)	90.01	54.32	41.29	79.93	63.21	61.48
LSTM	56.16	54.33	24.02	76.58	57.16	52.41
TD-LSTM	68.40	59.06	38.99	77.53	59.79	52.96
Flow-LSTM	97.06	**59.70**	**49.98**	97.50	**72.93**	**71.80**
Split-LSTM	**97.51**	56.45	46.65	**97.44**	72.02	70.25
ML-Split-LSTM	82.71	58.57	43.07	84.00	68.72	65.00

As we can see our models achieves the best mean validation accuracies and F1-scores, both for the original data set and the oversampled one. The lexicon-based models generally performs the worst, with the full lexicon-based model actually being achieving better performance than the one with handcrafted features. This is possibly due to the handcrafted model being designed for binary classification. For the remaining models we see a disparity in results across the two different training sets. For the oversampled dataset we achieve high accuracies and F1-scores, which is to be expected when having duplicate examples in both the training and evaluation set. The interesting observation here is that while this can happen to all the models, our models significantly outperform the baselines. For the original dataset the accuracies and F1-scores are also generally higher for our models, with TD-LSTM being close up there in accuracy as well. The learning curves of the models is shown in Figure 5.3

We can see in these graphs that the models converge very fast on the original dataset. This indicates that there is a strong bias because of the class imbalance and because an overfitting problem seems to happen very early on. For the oversampled data, the models have more stable learning and overall higher accuracies, but an overfitting problem still occurs.

5.3.2 Effect of Embeddings

The input feature vectors to the models plays a big role in their performance, thus it is important that we have high quality embeddings. Experimenting with getting the best embeddings has not been a major focus of this project, however we do wish to examine how much the dimensionality of the embeddings affect our models' performance. As we have a very large dataset, there exist many different

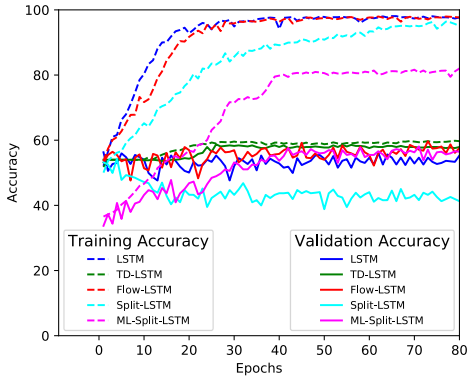


Figure 5.1: Model performance on the original data.

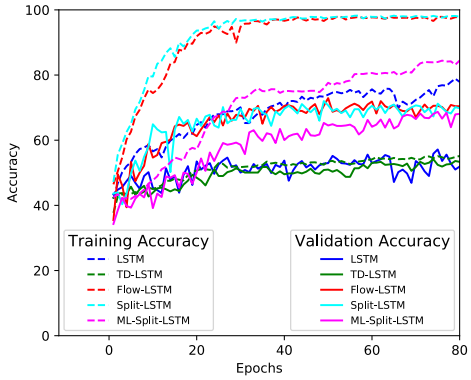


Figure 5.2: Model performance on the oversampled data.

Figure 5.3: Mean cross validation accuracies per epoch.

combinations and contexts that has to be modelled. This could mean that information is lost by choosing too few dimensions. Previous studies however has shown that after a certain number of dimensions, performance no longer improves that much [46]. As a result we evaluate our models using both 100, 200 and 300 dimensions for our embedding vectors. The performance on our models with different embeddings sizes can be seen in Figure 5.4.

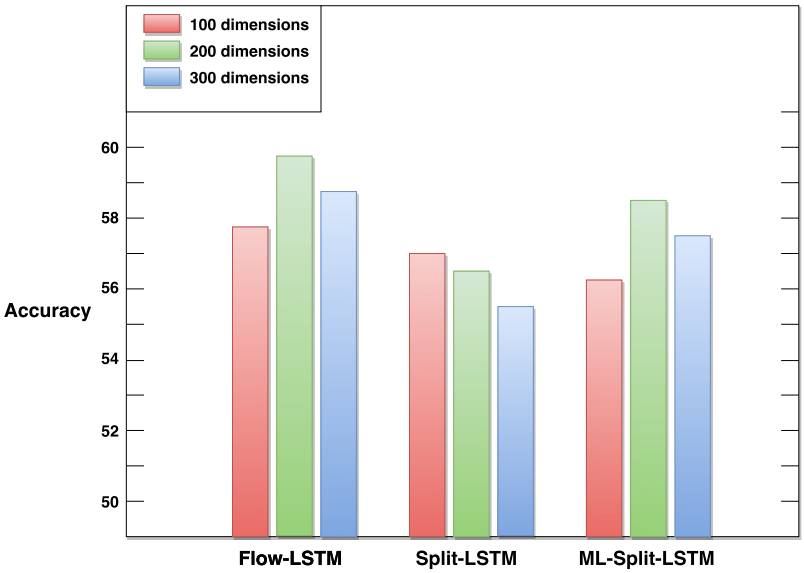


Figure 5.4: Comparison of the accuracy of Flow-LSTM and Split-LSTM using different embedding sizes.

This experiment is performed on the original dataset with no oversampling.

As we can see there is generally a performance gain to be had by using 200 dimensional vectors over 100 dimensional ones. 300 dimensions do not add any significant improvements, thus we choose to continue our experiments with 200 dimensional embeddings.

5.3.3 Model Overfitting

While we see reasonable scores in the previous experiment, we also see a very high training accuracy with a big gap to the validation accuracy, meaning that an overfitting problem occurs. To avoid this we introduce dropout into the models. As mentioned earlier, we experiment with both dropout on the input layer and inside the LSTM cell. Dropout on the input layer is especially interesting for the oversampled dataset. This should improve generalization greatly, as the oversampled examples will now be scrambled and most likely no longer be duplicates.

We achieve best results with both dropout values set to 0.25. We compare the performance of the resulting models to our models with no dropout, seen in Figure 5.7

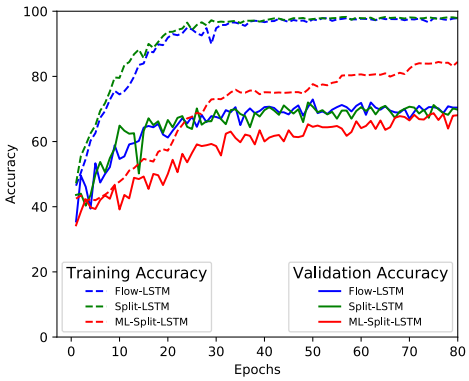


Figure 5.5: 80 epochs without dropout.

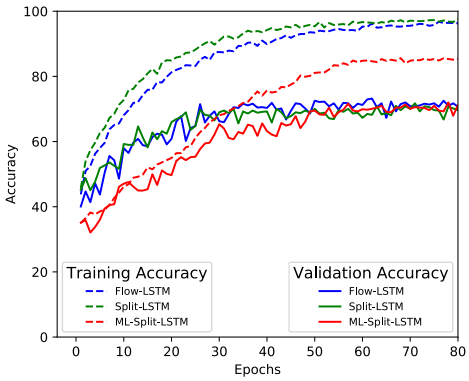


Figure 5.6: 80 epochs with dropout.

Figure 5.7: Mean cross validation accuracies per epoch.

These figures shows that dropout has a small effect on overfitting, but it is not a noteworthy difference. The problem seems to be in the class imbalance, for which dropout does not really help. Thus we choose to continue our experiments without dropout.

5.3.4 Test Results

We now run the models with their optimized hyperparameters on the test set. We present the resulting accuracy of each model on the test data set, as shown in in Table 5.2.

Table 5.2: Test results

Models	Original		Oversampling	
	Test	**F1**	**Test**	**F1**
Full Lexicon-based	35.33	29.93	35.33	29.92
Handcrafted Lexicon-based	29.33	20.93	29.33	20.93
Paragraph Vector (MLP)	54.67	**42.08**	44.00	36.05
LSTM	40.67	30.01	48.67	38.56
TD-LSTM	**58.00**	39.88	**54.66**	**43.62**
Flow-LSTM	47.33	33.07	53.33	36.18
Split-LSTM	53.33	36.51	54.00	37.98
ML-Split-LSTM	44.00	25.18	47.33	35.85

This shows that our trained models does not fit the test data very well. The lexicon-based methods still perform the worst, while the rest of the models are very mixed in their results. Overall it seems that TD-LSTM performs the best on this dataset. Across the original and oversampled dataset, we see a slight increase in accuracy for our models on the oversampled dataset. The same goes for the F1 score, which is higher for all models on the oversampled data. Still, the accuracies are equivalent to simply always predicting positive. This indicates that what the models have learned from the training set, does not apply to the test set at all. This is most likely a problem with the limited number of training examples we have. With such a limited number of training examples, it is simply not representative for our overall very large dataset. As such we see very little correspondence between the training and test set, and as a result over models have greatly overfitted the training data, resulting in less than optimal predictions on the test set. To analyse our hypothesis of there being a mismatch between the two sets, we plot the document embeddings of the datasets, using t-SNE for dimensionality reduction. This is visualized in Figure 5.8

This visualization does show, that there seem to be a difference between the two sets. The embeddings of the training set are grouped together a lot and does not fit a lot of other data. The test set is generally located separately from the training set, which could help explain the less than optimal predictions when training on this training set. In theory by having more training data, it should fit the test set better and result in better predictions. This visualization is of course not complete proof of our theory, as it only considers the relation of documents vectors, but it does indicate that there is a problem here.

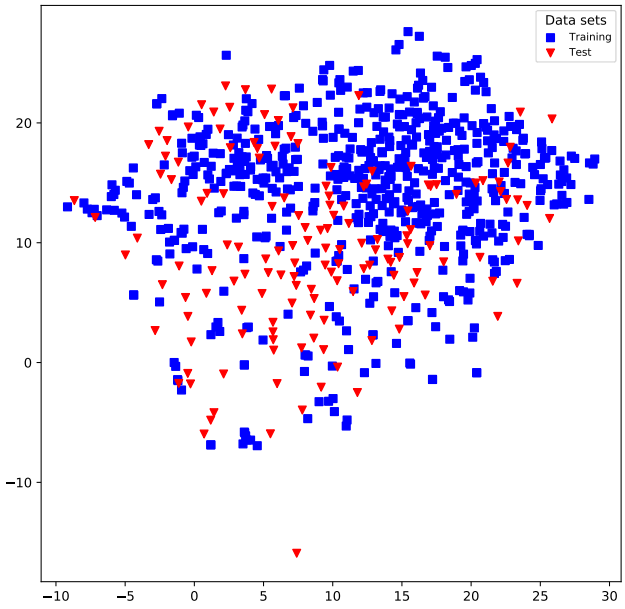


Figure 5.8: Document embeddings for our training and test set, reduced to two dimensions using t-SNE.

5.3.5 Generalisation

By only using labelled data for four supplements, we have made a major assumption that supplements are discussed in similar contexts and located within close proximity in the embedding space. To test this assumption, this experiment and evaluation metric analyses our model's accuracy on a previously unseen target. We wish to show that our model can generalize to other supplements and thus be used for target-dependent sentiment classification, given any supplement in the corpus.

With this in mind, perform the experiment of target-dependent sentiment analysis on the new target. Table 5.3 shows our results of running the models on training examples with the supplement calcium as the target, which has not been used for training.

Surprisingly this shows better results than on the test set. On this dataset out models significantly outperfom all the baselines. The lexical models remains least accurate, while the other models except ours, perform worse than just predicting positive. Our models achieve much better results, which indicates that they are effective on the right data. It is like that the training set fits this set better than the test set and thus we get better performance. These results also indicates that it is possible for the models to generalize to other supplements. However, calcium is very similar to vitamin D, for the contexts in which they are used. It would be interesting to evaluate a completely different supplement and see if we get the same results.

Table 5.3: Calcium test results

Models	Original		Oversampling	
	Test	**F1**	**Test**	**F1**
Full Lexicon-based	38.89	34.53	38.89	34.53
Handcrafted Lexicon-based	40.28	26.72	40.28	26.72
Paragraph Vector (MLP)	45.83	34.69	44.00	36.05
LSTM	43.06	34.85	44.44	31.51
TD-LSTM	50.00	22.22	50.00	29.14
Flow-LSTM	58.33	**49.36**	56.94	39.99
Split-LSTM	**65.27**	46.08	**59.72**	**48.55**
ML-Split-LSTM	58.33	39.21	52.78	46.53

5.3.6 Determining Relatedness to Cancer

As an additional experiment, we wish to examine if it is possible to classify, whether a target relates to cancer treatment or not. For training our models on this task, we have annotated each target post containing coffee with such a binary label, resulting in 231 labels. The division into a training and test data set is done based on the other training and test set. We have 185 training examples and 46 test examples, which are subsets of the overall training and test set respectively. The purpose of this classification is to improve the model, by only considering target instances which are related to the given type of context. For this experiment our models only differ by changing the output dimension of the softmax layer to two classes instead of three. ML-Split-LSTM is not evaluated here, as one one type of label is used. Our results can be seen in Table 5.4.

Table 5.4: Cancer relatedness results.

Model	**Test**	**F1**
Paragraph Vector (MLP)	69.57	51.65
LSTM	**78.26**	57.88
TD-LSTM	73.91	42.50
Flow-LSTM	67.39	56.52
Split-LSTM	76.09	**60.31**

Here we see much higher accuracies and F1-scores than on the previous sets. Surprisingly, the standard LSTM achieves the highest accuracy here. It is possible that because coffee in most cases does not relate to cancer, thus appearing in different contexts, makes target-dependence less important. Split-LSTM also achieves

good performance and seems to be the overall best model for our experiments.

There is most likely multiple reasons for the accuracies being much higher for this experiment. The first reason, is that it is a simpler problem when only two classes are considered instead of three, and only a single supplement is considered as well. This should make it easier for the models to fit the data.

This dataset is also very imbalanced with 73% "false" labels, meaning that a high accuracy can be achieved simply by predicting this class all the time. This does however not seem to be the case, as the models also achieves very good F1-scores and Split-LSTM also gets a higher accuracy.

In summary, we have seen that our models, especially Split-LSTM perform well on given datasets. On the final test set however, performance was not as good. It seems that with the limited amount of training data, the training set is not representative of our dataset, thus a disparity between the training and test set happens and the models cannot make accurate predictions. It does not look like it is a problem with our models, as they generally perform better than our baselines on all tasks. With this in mind, we conclude on this these in the following chapter.

Chapter 6

Conclusion

In Section 1.1.6 we stated the overall problem of this thesis along with a number of related questions. To answer this problem statement, we start by answering these questions.

What are the challenges of manually annotating forum threads?

The biggest challenge regarding the manual annotation of data has been to reach a sufficient amount alongside all the other thesis work. It is a time consuming task, and being only two annotators makes it take a long time to produce a meaningful amount of labels. The way we designed our models, also required both target and general labels to be made. This meant that for each training example, 6 labels had to be produced. As a result, with around 5000 total labels, after splitting into train and test sets, there are only 613 training examples.

Another challenge is the agreement between annotators, as shown by our agreement study in Section 3.2. The forum posts we have labelled has proven to often be rather long and contain a number of mixed opinions. With many of these opinions, it becomes a difficult task to determine, what is the overall polarity of the post. Determining coffee's relatedness to cancer however, was much easier and we achieved full agreement in our study. Here the distinction between the binary classes were always very clear, but that was not the case for the polarity classes.

How can word and document embedding vectors be successfully combined as input to one model?

We experimented with two different ways to input two different classes of embeddings into one model. With Flow-LSTM we made the assumption that word and document embeddings trained using the same neural model exist in the same vector space. The model is nearly identical to TD-LSTM with the idea being, that

adding document embeddings as a direct extension of word embeddings can improve performance. This is difficult to conclude from the experiments as both models differently on different tasks, and the results are affected by the disparity between the training and test set. However, based on the cross-validation results, we conclude that there is a lot of potential given more representative data. On the other hand, Split-LSTM processed the document and word embeddings separately and got more consistent results on unknown data, indicating that this is a better solution, although the results were still not very good on the test set.

How do we optimize the models with the purpose of target dependent sentiment classification?

Like with TD-LSTM we experimented with processing both the preceding and following context of a target, with the target being the end of each sequence. We did this on both word and document level to capture the notion of both target word and target post. We conclude this is an affective method to take targets into account, as the performance is generally much better than the standard LSTM.

How can document sentiment labels be incorporated as a normalizer?

With ML-Split-LSTM we experimented with combining both types of labels into one prediction, and using multiple loss functions, making it a multi-input, multi-output network. We thus use the models ability to predict the combined general label for the surrounding context to the final loss function. This has generally yielded the worst results of all our three models. It is however difficult to evaluate whether this is a problem with the design, or mostly due to limited data, as this is the most complex of the models, thus requiring the most data.

Based on answers to the aforementioned questions, we can now answer our problem statement.

How can the hierarchical and sequential structures of a forum thread be modelled in a target dependent way using Long Short-term Memory networks?

We have shown three possible architectures of how continuous, multi-level features can be used with LSTM networks to model the structure of a forum thread. We have shown that considering the global context in addition to the local context has potential if given more training data. The best way to model the hierarchy is not completely clear from our results, but it seems that keeping the context separate and then learning a combination of them, performs the best when generalising to unknown data.

Chapter 7

Future Work

In this thesis we have addressed a wide amount of issues, but many changes and improvements could still be made. In this chapter we give an overview of future work that could be considered for improving target-dependent sentiment analysis in forum threads.

7.1 Additional Training Data

As we could see in our experiments, we did not have nearly enough labelled data to fully evaluate our models. With such a large dataset, we need more labelled training examples to have a training set that is representative for our data. As such, we consider the most important next step is to annotate more data. As discussed earlier there are multiple ways to go about this. As we now have more domain knowledge and have set up rules for the annotation process, after evaluating an agreement study, it makes sense that we should continue the annotation. On the other hand, we have shown, that it is a very time consuming process, and for only two people it takes a very large amount of time to produce a meaningful number of labels. Another option is to crowd-source the annotation on an online platform such as Amazon's Mechanical Turk. This will produce a much larger quantity of labels, but likely with reduced quality, as we have shown that agreement is difficult to achieve for this domain.

7.2 Active Learning

In our annotation process we have annotated random threads with posts containing our supplements. This has resulting in an uneven class distribution in our datasets, where negative labels are largely underrepresented. Now that classifiers have been trained, an idea is to use them to select the posts to label. One option to balance the datasets, is to return posts the models have predicted to be of the

underrepresented class, evaluate the results and retrain the models. Another is to query by uncertainty sampling, evaluating predictions where the models have low confidence about the correct class. Another method is to assume that documents embeddings close together also has the same class. Predicting the nearest neighbour target posts of a target posts, to be the same polarity and then querying the user about the correctness, could be another way to obtain the desired outputs at new data points. This can also be used for underrepresented classes in particular, to learn more examples of these faster.

7.3 Scaling the Loss Function

We attempted to deal with the problem of unbalanced classes, by oversampling the class to the level of the others. This resulted in only a small improvement on predicting on the test data. An alternative to this method is loss scaling. In our implementation, the loss function considers all classes to have equal importance, however scaling the loss functions output dependent on the predicted class, thereby assigning more importance to the underrepresented class, could yield higher probability for predicting this class. This could for instance be done by weighting the cross-entropy error for each class or by using a larger learning rate when the underrepresented class the underrepresented class is incorrectly predicted. Another idea is to incorporate the bias of the annotator as a feature of the model, by comparing the annotated class distributions of the annotates and finding a pattern which can be used for the loss scaling.

7.4 Create Better Input Data

Throughout the project a lot of time has been spent on processing the raw data and remove various errors. Most of these problems is a result of the data having been scraped without considering some of the more intricate differences between the sources. Now that we have knowledge of these difference, it could be worth looking into scraping the data again, taking into account these observations when designing the scraper. For instance, the forums which allows the users to create signatures, introduce a lot of noise in the text, which is difficult to automatically remove when it has been scraped as part of the post. By handling this when scraping the data instead, where there are tags that define what text to discard, could clean up a lot of the data.

7.5 Normalizing the Embeddings for Target Synonyms

We consider only four supplements, but also all their synonyms. When these all have different embedding vector representation, it can be argued that we have as

many supplements as there are labelled synonyms, but only very few on some of them. Normalizing the embeddings of supplements and their synonyms to one embedding per supplements could result in better vectors and also be more in line with how they have been annotated.

Bibliography

[1] Long Jiang et al. "Target-dependent twitter sentiment classification". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics. 2011, pp. 151–160.

[2] Li Dong et al. "Adaptive Recursive Neural Network for Target-dependent Twitter Sentiment Classification." In: *ACL (2)*. 2014, pp. 49–54.

[3] Duy-Tin Vo and Yue Zhang. "Target-Dependent Twitter Sentiment Classification with Rich Automatic Features." In: *IJCAI*. 2015, pp. 1347–1353.

[4] Duyu Tang et al. "Effective LSTMs for Target-Dependent Sentiment Classification". In: *arXiv preprint arXiv:1512.01100* (2015).

[5] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[6] Shalini Ghosh et al. "Contextual LSTM (CLSTM) models for Large scale NLP tasks". In: *arXiv preprint arXiv:1602.06291* (2016).

[7] Richard Socher et al. "Recursive deep models for semantic compositionality over a sentiment treebank". In: *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*. Vol. 1631. Citeseer. 2013, p. 1642.

[8] James Hong and Michael Fang. *Sentiment analysis with deeply learned distributed representations of variable length texts*. Tech. rep. Technical report, Stanford University, 2015.

[9] Charu C Aggarwal and ChengXiang Zhai. *Mining text data*. Springer Science & Business Media, 2012.

[10] Peter D Turney. "Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews". In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, pp. 417–424.

[11] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[12] Quoc V Le and Tomas Mikolov. "Distributed Representations of Sentences and Documents." In: *ICML*. Vol. 14. 2014, pp. 1188–1196.

[13] Tomas Mikolov et al. "Recurrent neural network based language model." In: *Interspeech*. Vol. 2. 2010, p. 3.

[14] Duyu Tang, Bing Qin, and Ting Liu. "Document Modeling with Gated Recurrent Neural Network for Sentiment Classification." In: *EMNLP*. 2015, pp. 1422–1432.

[15] Minlie Huang, Yujie Cao, and Chao Dong. "Modeling rich contexts for sentiment classification with lstm". In: *arXiv preprint arXiv:1605.01478* (2016).

[16] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. "Sentiment analysis algorithms and applications: A survey". In: *Ain Shams Engineering Journal* 5.4 (2014), pp. 1093–1113.

[17] Bo Pang and Lillian Lee. "A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts". In: *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics. 2004, p. 271.

[18] Bo Pang and Lillian Lee. "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales". In: *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2005, pp. 115–124.

[19] Yoon Kim. "Convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1408.5882* (2014).

[20] Kai Sheng Tai, Richard Socher, and Christopher D Manning. "Improved semantic representations from tree-structured long short-term memory networks". In: *arXiv preprint arXiv:1503.00075* (2015).

[21] Hao Wang et al. "A system for real-time twitter sentiment analysis of 2012 us presidential election cycle". In: *Proceedings of the ACL 2012 System Demonstrations*. Association for Computational Linguistics. 2012, pp. 115–120.

[22] WebMD. *About RxList*. `http://www.rxlist.com/script/main/art.asp?articlekey=64467`. Accessed: 28-04-2017.

[23] Joan M Lappe et al. "Vitamin D and calcium supplementation reduces cancer risk: results of a randomized trial". In: *The American journal of clinical nutrition* 85.6 (2007), pp. 1586–1591.

[24] Savita Bisht et al. "Polymeric nanoparticle-encapsulated curcumin (" nanocurcumin"): a novel strategy for human cancer therapy". In: *Journal of nanobiotechnology* 5.1 (2007), p. 3.

[25] Preetha Anand et al. "Curcumin and cancer: an "old-age" disease with an "age-old" solution". In: *Cancer letters* 267.1 (2008), pp. 133–164.

[26] Mia Hashibe et al. "Marijuana use and the risk of lung and upper aerodigestive tract cancers: results of a population-based case-control study". In: *Cancer Epidemiology and Prevention Biomarkers* 15.10 (2006), pp. 1829–1834.

[27] *WordNet Search - 3.1*. `http://wordnetweb.princeton.edu/perl/webwn`. Accessed: 05-05-2017.

[28] Jacob Cohen. "A coefficient of agreement for nominal scales". In: *Educational and psychological measurement* 20.1 (1960), pp. 37–46.

[29] Mary L McHugh. "Interrater reliability: the kappa statistic". In: *Biochemia medica* 22.3 (2012), pp. 276–282.

[30] Zachary C Lipton, John Berkowitz, and Charles Elkan. "A critical review of recurrent neural networks for sequence learning". In: *arXiv preprint arXiv:1506.00019* (2015).

[31] Alex Graves. "Supervised sequence labelling". In: *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, 2012, pp. 5–13.

[32] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. DTIC Document, 1985.

[33] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *arXiv preprint arXiv:1301.3781* (2013).

[34] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `http://tensorflow.org/`.

[35] Paul J Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[36] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[37] Sepp Hochreiter et al. *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. 2001.

[38] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." In: *ICML (3)* 28 (2013), pp. 1310–1318.

[39] Klaus Greff et al. "LSTM: A search space odyssey". In: *IEEE transactions on neural networks and learning systems* (2016).

[40] Arzucan Özgür, Levent Özgür, and Tunga Güngör. "Text categorization with class-based and corpus-based keyword selection". In: *International Symposium on Computer and Information Sciences*. Springer. 2005, pp. 606–615.

[41] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. "Recognizing contextual polarity in phrase-level sentiment analysis". In: *Proceedings of the conference on human language technology and empirical methods in natural language processing*. Association for Computational Linguistics. 2005, pp. 347–354.

[42] Nitish Srivastava et al. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[43] Yarin Gal and Zoubin Ghahramani. "A theoretically grounded application of dropout in recurrent neural networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 1019–1027.

[44] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* 4.2 (2012).

[45] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[46] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global Vectors for Word Representation." In: *EMNLP*. Vol. 14. 2014, pp. 1532–1543.

Appendix A

Data Analysis

A.1 Supplement Occurrences Analysis

The analysis shown in Table A.1 is conducted on the lower cased supplement synonyms and posts, thus resulting in specific cased words, is not correct analysed, e.g. the most mentioned synonym is "same", referring to s-adenosylmethionine (also known as SAMe) which is a artificially produced form of a chemical that occurs naturally in the body. However, the analysis can not distinguish between the chemical and the normal occurring word "same" as in, "I have the same diagnosis".

A.2 Annotator Agreement Study

The posts with disagreeing target labels is analysis in this section, to focus out attention to the challenges of annotating target labels. For each such challenge, we reach an agreement and decide rule which we will follow in the future labelling. Note, there was a total of 21 posts, but we only describe the x ones who is an example of each individual challenge. Mentions of the target synonyms in each post is highlighted with bold.

Challenge 1 in Example A.1: The content of this post includes a quote of a previous post, which is positive towards curcumin. However, the content written by the writer of the post has a neutral opinion towards curcumin. **Rule 1:** We chose to discard all content in a post not being written by the user posting the post.

Example A.1 (Target: curcumin - peter: positive - Emil: neutral - Agreed: neutral)
" Originally Posted by tintomatoesSorry but this is not a fact. You can survive and reduc prostate cancer with ONLY diet and lifestyle changes - medical fact! And **curcumin** happens to be a very effective agent in but not well absorbed so I would take a extract ideally mixed with Phosphatidylcholine like the famous brand but you dont need to buy that brand as many others have made the

Table A.1: The most mentioned supplements using all of their synonyms, and descriptions of why they were chosen of not, as targets.

Supplement	# posts	Description
Cannabis	79.576	Chosen: alternative supplement for cancer treatment, that many people are interested in, but has small amount of research.
Calcium	65.059	Not chosen: the synonyms of chemical elements are to general, and thus not always in the context as a supplement.
Wine	61.721	Not chosen: the general synonym "alcohol".
Coffee	44.799	Chose: contrast to cannabis since not always related to cancer.
Beer	42.012	Not chosen: Same problem as with "wine".
Vitamin D	40.173	Chosen: normal supplement for traditional cancer treatment.
Cocoa	35.142	Not chosen: it is similar to "coffee".
Apple	28.446	Not chosen: it is similar to "coffee".
Lemon	26.472	Not chosen: it is similar to "coffee".
Iodine	26.102	Not chosen: it is similar to "Calcium".
Magnesium	26.029	Not chosen: it is similar to "Calcium".
Iron	24.521	Not chosen: it is similar to "Calcium".
Curcumin	24.486	Chosen: alternative supplement for cancer treatment, that many people are interested in, but has small amount of research.

> mix much cheaper - pm me if you need more infoFor dosage of **curcumin** ref /tintomatoes,I have to totally agree with Fairwind on this one. There are NO peer reviewed studies from accredited sources that show that cancer of any type can be cured or controlled by diet modification alone. To state this as fact is to give false hope. A word of advice, your post comes close to a sales pitch in my opinion and that is generally frowned on here. There IS a forum here which is devoted to alternative therapies. I suggest you try posting there. "

Challenge 2 in Example A.2: We have disagreement whether it neutral or positive that a user states she takes/drinks a supplement daily, such as coffee or vitamin D. **Rule 2:** We decide it is always a positive opinion, that a user takes/drinks a supplement daily.

> **Example A.2 (Target: coffee - peter: positive - Emil: neutral - Agreed: positive)** " I also take my multi-vitamin then, so that if I burp vitamin breath, no one will know.. ROFL!! I take once-a-month Actonel..sigh..have to take it in the morning, on an empty stomach, with LOTS of water, and then nothing for 30 minutes..sounds easy? You DO NOT want to see me in the morning without my

COFFEE!! As far as the OTC stuff, ask your trial proctors..it may be different, since they are watching you.. Hugs, Kathi ”

Challenge 3 in Example ??: Contra challenge 2, the action of having coffee with some one is another challenge, regarding coffee. **Rule 3:** The action of having coffee with is not related to the supplement it self, thus we decide it is always a neutral opinion towards coffee.

Example A.3 (Target: coffee - peter: positive - Emil: neutral - Agreed: neutral) “ I’m trying to figure out if this means I never get another **coffee** with Bizezgrrl! ;-) I was also a workaholic at work, but have been pretty much a nothing-aholic sluggard since leaving. There are days when I ask myself whether that was the best idea, but when I start running possible scenarios, the answer almost always comes up, "Yes, it was a good idea. ”

Challenge 4 in Example ??: It is difficult to label a target if the post contains multiple different opinion. And in this post the user takes vitamin D daily, witch by rule 2 is positive, however it has not helped improved here diagnose, thus resulting in a neutral label. **Rule 4:** The annotator should notice all the different opinions in the post, and annotate the polarity the majority of the opinions contain. Of course there is many different cases of polarity mixes, but there is a uniform distribution of polarities the target should always be labelled as neutral.

Example A.4 (Target: vitamin D - peter: positive - Emil: neutral - Agreed: neutral) “ Gatekeeper, With dedication to a exercise program over 10 years, which includes daily walking,yoga, Pilates reformer, ballet and jazz for fun; with a healthy nutritional plan plus **Vitamin D**, calcium and other vitamins and minerals, I have yet to come out of a -3 range hips and spine. I know of many people with osteoporosis that also are very dedicated to exercise and healthy food; which may be easier living in a sunny climate, but I have not heard of anyone making such a drastic change with bone loss. Sandi ”

Challenge 5 in Example ??: If the cancer treatment makes a patients taste buds change to for instance coffee, what is the polarity towards coffee as a target then. **Rule 1:** If the traditional cancer treatment alters a patients taste buds such that the user dislikes coffee, it should always be labelled as neutral, because the negative opinion is not related to the supplement but to the traditional treatment such a chemotherapy.

Example A.5 (Target: coffee - peter: negative - Emil: neutral - Agreed: neutral) " 12 years ago when I had my treatment for SCC at base of tongue, mt taste buds were all messed including the sense of sweetness. It took several years for most flavors to come back but even today there are flavors still not right. For years I had to dump sugar into my coffee for it not to taste better; now, all of a sudden it is too sweet and having to back off the amount of sugar. Chocolate has never recovered, it taste bitter and burnt. If you had radiation treatment, I am sure your medical team has told you all radiated area will continue to change over the years; some good, most bad. Damaged tissues in throat is reason I am going in for full laryngectomy surgery the 22nd to prevent aspiration pneumonia. Roy, SCC Survivo "

A.3 Experiments

Table A.2: 5-fold cross validation results of the four standard classification models utilizing 200 dimension paragraph vectors.

Models	Original training data			Oversampling training data		
	Train	**Val**	**F1 macro**	**Train**	**Val**	**F1 macro**
MaxEnt	64.15	**58.57**	38.46	62.52	53.36	49.46
GaussianNB	61.26	51.07	**42.95**	57.88	50.59	48.96
MLP	90.01	54.32	41.29	79.93	**63.21**	**61.48**
SVM	54.00	54.00	23.36	43.50	43.50	20.17