Daan Kolthof

# Recognizing and Handling Negations in Machine Learning

TU Delft

# Recognizing and Handling Negations in Machine Learning

Daan Kolthof

August 23, 2018

In several machine learning problems, a relatively small subproblem is present in which combinations of (negating) objects or structures result in a negation or otherwise other classification compared to when these (negating) objects are not present. To be more specific, a variant of the XOR problem is present in a small amount of objects in these classification problems. Examples of this could be negating words in textual sentiment classification or the presence of sarcasm when one wants to determine seriousness in speech. As negations are usually present in a small part of much larger datasets, it is important to recognize these relatively rare negation structures within objects' data. Correctly recognizion and handling negations could improve overall classification performance in machine learning problems that inhibit negations in some of their dataset objects. To lay the groundwork for solving these problems, the subproblem of recognizing negation words in sentiment classification is solved by employing a word embedding neural network to recognize text structure and to correctly classify these negations while at the same time this neural network is used to classify complete sentences in the problem of sentiment classification.

# Contents


**1 Introduction** **6**
- 1.1 Negation recognition . . . 6
  - 1.1.1 Two-class classification . . . 6
  - 1.1.2 Negation recognition and the XOR problem . . . 7
  - 1.1.3 Negations within datasets . . . 7
- 1.2 Practical example: Sentiment Classification . . . 7
  - 1.2.1 Document classification . . . 8
  - 1.2.2 Sentiment classification . . . 8
  - 1.2.3 Sentiment negation handling . . . 9
  - 1.2.4 Short text classification . . . 11
- 1.3 Practical example: Sarcasm in speech recognition . . . 11
- 1.4 Outline . . . 12

**2 Background** **13**
- 2.1 $n$-grams . . . 13
- 2.2 Models for document classification . . . 13
- 2.3 Neural networks and language processing . . . 15
- 2.4 Language processing through word embedding . . . 16
- 2.5 Combining neural networks and word embedding . . . 17
- 2.6 The attention mechanism in neural networks . . . 17
- 2.7 Imbalanced datasets and sampling . . . 18
  - 2.7.1 Undersampling of the majority type . . . 19
  - 2.7.2 Oversampling of the minority type . . . 19
  - 2.7.3 Focused oversampling . . . 19

**3 Experimental setup** **21**
- 3.1 Classification method: Word Embedding Neural Network . . . 21
- 3.2 Dataset . . . 22
  - 3.2.1 Negation recognition objects within the dataset . . . 23
- 3.3 Neural network design . . . 23
- 3.4 Negation recognition and handling . . . 24
  - 3.4.1 Training on a slimmed down dataset . . . 24
  - 3.4.2 Training on the full dataset . . . 25
- 3.5 Word embeddings . . . 26
- 3.6 Combining partial classifications . . . 26
  - 3.6.1 Combining method: Averaging subset $n$-gram classifications . . . 26
  - 3.6.2 Combining method: Averaging the minimum and maximum subset classifications . . . 27
  - 3.6.3 Combining method: Taking the strongest of the minimum and maximum values . . . 27
  - 3.6.4 Combining method: Taking the (first) negation subset classification . . . 27
  - 3.6.5 Experimental dataset setups . . . 27

**4 Results** **29**
4.1 Neural network design . . . . . 29
4.1.1 Embedding dimension . . . . . 33
4.1.2 Nodes in the first hidden layer . . . . . 33
4.1.3 Second hidden layer . . . . . 33
4.1.4 Final design . . . . . 33
4.1.5 Influence of the number of epochs on classification performance . 34
4.2 Slimmed down dataset . . . . . 35
4.2.1 Results . . . . . 36
4.3 Full dataset . . . . . 36
4.3.1 Baseline performance . . . . . 36
4.3.2 Adding artificial objects to the dataset . . . . . 38
4.3.3 Generalization when adding artificial objects to the dataset . . . 39
4.4 Word embeddings . . . . . 40
4.4.1 Embedding of sentiment . . . . . 40
4.5 Combining partial classifications . . . . . 42

**5 Conclusion** **45**
5.1 What has been done? . . . . . 45
5.2 What can be concluded? . . . . . 46
5.3 Future work . . . . . 46

# 1 Introduction

## 1.1 Negation recognition

Within classification problems, there exists a subproblem that is called negation recognition (and handling). This problem is characterized by classification objects containing a structure within these objects that, when present, negate or reverse the meaning (classification) of this object. So if this negation structure is present, the meaning (and thus resulting classification) of an object is negated compared to when this structure is not present in said object. An example of a negation within the machine learning problem of textual sentiment classification (determining whether a text sentence is of positive or negative sentiment) is presented in Table 1.

Table 1: An example of a negation within the problem of textual sentiment classification (determining whether a text sentence is of positive or negative sentiment).

| **Object (sentence)** | **Desired classification** |
|---|---|
| "The weather is so good today." | Positive |
| "The weather is not good today." | Negative |

The example given in Table 1 clearly shows how negations within a text complicate the classification of said texts. The negation structure, in this case the word or words that cause the classification result to be negated, is easily recognized: the word 'not' followed by the word 'good' in the second object indicate a negation of a positive term, thus leading to a negative classification.

As the more practical experiments in this research will be focused on the problem of negation recognition, as well as complete sentiment classification, using a word embedding neural network, it is important to first clarify the relation between negation recognition and other machine learning notions and problems, as well as the drawbacks of existing machine learning methods with respect to negation recognition. In these following subsections, the relations between the problem of negation recognition and other machine learning concepts, as well as the appearance of negations in real-world datasets are discussed.

### 1.1.1 Two-class classification

Not necessarily connected to any negation recognition problem, two-class classification describes classification problems and datasets where the objects within these datasets can be labelled as one of two classes.
In relation to the problem of negation recognition and handling, it is relatively straightforward to handle negations (if present) in two-class classification problems. This due to the fact that it is easy to negate or 'flip' the predicted class of an object, as there are only two classes to choose from. When an object is predicted to be of a certain class, this class is simply 'flipped' when a negation is detected. When classification

problems are attempted to be solved with more than two possible classes to classify objects as, negation recognition is much less straightforward as there is no easy way to negative the classification label of an object, as there are more possible alternative labels for objects containing a negation.

### 1.1.2 Negation recognition and the XOR problem

A relatively simple problem that will form the base for the more complex problem of negation recognition and handling is the XOR problem. The XOR-problem is often mentioned when one wants to make a distinction between linear and non-linear classifiers[21]. Table 4 presents the XOR-problem, with its possible inputs and expected outputs. Figure 1 displays these values in a 2D space, displaying the relation between the XOR problem and the problem of negation recognition and handling and immediately making it clear why the objects in this problem cannot be separated by a linear classifier.

### 1.1.3 Negations within datasets

As mentioned earlier, recognizing and handling negations is often a subproblem of a larger machine learning problem. The reason negation recognition is seen as a subproblem of said larger machine learning problem is that often the datasets relating to certain machine learning problems contain only a relatively small amount of objects that actually contain a negation.
When the objective of training and testing a classifier is to maximize the achieved classification performance (during training), these negation objects will often not be deemed important enough to significantly influence the training of said classifiers. This due the fact that learning to recognize and handle these negations will have a high cost (much larger training time, increasing error on non-negation objects, etc), which usually does not outweigh the relatively small increase in overall accuracy.
Because of the fact that negation recognition and handling is often a subproblem that has to be learned from a subset of an entire machine learning dataset, the classifiers that perform well on this subproblem are often not the same classifiers that do perform the best on the much larger, original, machine learning problem. An example of the use of a special classifier that aims to maximize the classification performance of negation objects is that neural network that is discussed as a result of this research.

Now that the notion of negation recognition and the relation between a negation recognition problem and other, more general machine learning problems is explained, some practical examples of negations within datasets are explored.

## 1.2 Practical example: Sentiment Classification

This problem was briefly discussed earlier in this section and will be thoroughly analysed when the actual experiments of this research are done. Before the problem of

negation recognition within textual sentiment classification is introduced, some problem definitions of larger and more general machine learning problems are introduced. The problems that are introduced are, in order of appearance: document classification, (textual) sentiment classification, negation recognition within sentiment classification and the notion of short text classification.

### 1.2.1 Document classification

Document classification is the task of assigning a document to one or more document categories, also called document classes. Classification of a document is done by determining the probability distribution of all possible classes based on the contents of said object. As this classification result entails a probability distribution, it must mean that the sum of all class probabilities $p(c_i|D)$ must be 1, as formulated in equation (1) (with $D$ the document, $c_i$ each of the classes and $C$ the total number of classes).

$$\sum_{i=1}^{C} p(c_i|D) = 1 \tag{1}$$

The definition of a document can differ as well, as multiple sources of data can be considered 'documents'. These sources range from textual data to images or audio. These different sources of data require different methods of processing to perform classification. This research is focussed on textual data, which means a document can be described as an ordered set of words (sometimes also called terms). By following this definition, a document $D$ can be formulated as:

$$D = \{w_1, w_2, ..., w_N - 1, w_N\} \tag{2}$$

In this formulation, $N$ denotes the size of document $D$, in number of words (also written as $|D|$).

### 1.2.2 Sentiment classification

Sentiment classification, sometimes also known as sentiment analysis, is a case of document classification. Textual sentiment classification refers to the use of natural language processing methods to classify the sentiment of textual data. The most straightforward sentiment classification is classification of documents as one of two categories (also called classes), often the 'positive' and 'negative' classes (or something similar). Sentiment classification with more than two classes can be done as well, in which the classes usually indicate the strength of positivity and negativity (or the strength within some other range).
To give a practical example of sentiment classification, consider the tweets displayed in Table 2. For a human reader, it should instantly be clear which of the example objects display a positive sentiment and which ones display a negative sentiment.

Table 2: An example of sentiment classification. The objects in this example are tweets, classified as positive or negative.

| **Object** | **Classification** |
|---|---|
| "back from school. I fell down 3 flights of stairs today... you could see the tissue in my leg. im sore all over" | Negative |
| "now ive got a bruise on my hand cause i flung and hit the table when i put them on." | Negative |
| "i have a new brother wow im so happy XD" | Positive |

Within the problem of sentiment classification, there are a few special situations, in which objects have certain semantic characteristics that influence the classification of said objects. Often, a more in-depth knowledge of the used language is required to grasp these characteristics and to correctly classify the objects. Some of these semantic characteristics are shown here, including an example sentence:

- "Monday mornings are the best!" (Possible sarcasm)
- "Bad weather only occurs very occasionally" (Adverb influences the sentiment)
- "I do not like basketball" (Negation of positive term)
- "The weather is not bad today" (Negation of negative term)

### 1.2.3 Sentiment negation handling

As demonstrated in some of the examples presented earlier, the case of recognizing negations of positive and negative words is a special subproblem within sentiment classification and is a less straightforward problem compared to general sentiment classification.
The problem of negation handling, also called negation recognition, is defined as recognizing combinations of words that, together, have a different classification compared to these words separately. To be more specific, the combinations of negating words and positive or negative words. Negations, which contain a negation word and a positive or negative word need to be recognized and classified correctly. An example of a negation would be the text *"not cool"*. This text should be classified as a negative sentence, while the word *"cool"* with a neutral word (a non-negating word) in front of it should be classified as a sentence with a positive sentiment.
A complete overview of all possible combinations of negating words, positive and negative words, and neutral words and their respective classifications is displayed in Table 3.

Table 3: Description of different objects in the negation recognition problem and their desired classifications.

| **Object description** | **Example sentence** | **Desired classification** |
|---|---|---|
| negating word + positive word | "not happy" | Negative |
| negating word + negative word | "not bad" | Positive |
| neutral word + positive word | "is awesome" | Positive |
| neutral word + negative word | "so boring" | Negative |

The different objects in the negation recognition problem can be mapped to the 4 different objects in the XOR problem(and the other way around), as shown in Table 4. The objects in the negation recognition problem are not linearly seperable, as shown in Figure 1. This means that there is no linear classifier that is capable of correctly classifying the objects in the negation recognition problem.

Table 4: Objects in the negation recognition problem and their relation to objects in the XOR problem.

| **Negation recognition object** | **Negation recognition classification** | **XOR object** | **XOR classification** |
|---|---|---|---|
| negating word + positive word | Negative | (1, 1) | 0 |
| negating word + negative word | Positive | (1, 0) | 1 |
| non-negating word + positive word | Positive | (0, 1) | 1 |
| non-negation word + negative word | Negative | (0, 0) | 0 |

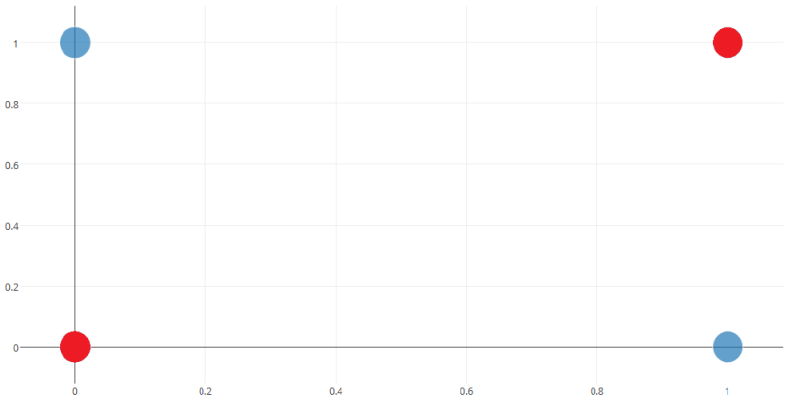


Figure 1: The four objects that represent the negation recognition problem(as well as the XOR problem). The red objects represent objects of negative sentiment(0) and blue objects represent positive sentiment(1). It can be seen that these objects are not linearly separable.

### 1.2.4 Short text classification

Short text classification can be seen as a (special) case of document classification. This case of document classification is encountered when the objects to classify are relatively small texts. For certain natural language processing problems, having short texts means different classification methods have to be used[26]. Some of these classification methods are tailored specifically to small texts, or are needed because the more general methods suffer from the relatively small amount of information in short texts.

## 1.3 Practical example: Sarcasm in speech recognition

Another interesting machine learning problem is the problem of speech recognition. Methods and classifiers related to this problem are focussed on interpreting human speech and translating it to a format that is understandable, and can be processed by, computers[25][12]. Within the area of speech recognition, more specifically, the problems of classifying seriousness or sentiment, certain types of negations exist. One example of these negations would be the presence of sarcasm in speech data where a classifier aims to recognize the sentiment of this speech data.

In this research, the focus lies on textual data and the relevant negation problems that arise. However, the versatility of machine learning models could mean that any insights gained in text classification problems could be used for speech recognition as well. This notion of using models previously constructed for image and text related classification problems in speech recognition has been done before, with results that are often an improvement compared to earlier models[17].

## 1.4 Outline

Now that the notion of negation recognition is explained, including some real-world examples, an overview of the objectives and research questions addressed in this research is presented. The objectives of this research are as follows, these objectives are all related to the problem of negation recognition within textual sentiment classification:

- Finding out the best classifier design for use for negation recognition within the context of sentiment classification;
- Developing strategies for preprocessing the dataset that is used during training and testing of the chosen classifier;
- Finding out what can be learned about individual words when training the classifier: do the learned word representations say anything about the meanings of said words?
- To propose techniques for combining classifications of (fixed-size) subsets of input sentences, providing ways to classify complete input sentences.

These research objectives will lead to experiments that are done on, or utilizing, a word embedding neural network. The notions of neural networks and word embeddings, as well as relevant background methods and techniques that lead up to them, will be introduced in the next chapter of this report.

This report is outlined as follows. Chapter 2 will provide background knowledge about the machine learning concepts that are relevant to negation recognition, document classification, neural networks and/or any of the objectives listed above. Chapter 3 will provide an overview of the setup of all experiments that are constructed to perform the research objectives, it will provide information on how to setup datasets and classifiers, and how to interpret the result data. Chapter 4 will contain the actual results of said experiments, and will provide explanations of these results and their meaning in the broader context of negation recognition. Chapter 5 will reiterate over the research that is done, and will provide a concluding summary of the research, as well as directions for future research that follow from the findings in this report.

# 2 Background

This section will contain background information relevant to the experiments that will be done and the conclusions that will be drawn based on these experiments. The theory presented here is linked to the natural language processing problems of document classification in general, sentiment classification and to negation recognition, as well as the methods used to solve these problems, which includes the technique of neural networks, word embedding and the attention mechanism. Furthermore theory on balancing datasets is presented, as the problem of imbalanced datasets will be encountered in the later experiments that are done and presented in the next section.

## 2.1 $n$-grams

A relatively common structure that is used in the area of natural language processing, more specifically in the area of text processing, is the $n$-gram structure. This structure can be seen as a string consisting of $n$ terms (words). When an input text contains more than $n$ words, the $n$-grams are taken by taking all sets of consecutive words such that each set contains $n$ words. An example of creating these $n$-grams is shown in Table 5.

Table 5: An example of creating a collection of $n$-grams

| | |
|---|---|
| **Original text** | "The quick brown fox jumps over the lazy dog" |
| **2-gram** | {"The quick", "quick brown", "brown fox", "fox jumps", "jumps over", "over the", "the lazy", "lazy dog"} |
| **3-gram** | {"The quick brown", "quick brown fox", "brown fox jumps", "fox jumps over", "jumps over the", "over the lazy", "the lazy dog"} |

As clarified in the example shown in Table 5, the set of all $n$-grams obtained from a sentence, text or document depends on the parameter $n$ that is chosen. A document consisting of $l$ terms will have $l - n + 1$ different $n$-grams (given $n \leq l$).

## 2.2 Models for document classification

One model for document classification is the **bag-of-words model**. In the bag-of-words model, a document is modelled as an unordered collection of words. This modelling can be done in two slightly different ways: encode the document by counting the frequency of each word in the document or encode the document by simply keeping check whether a word appears in the document or not (without keeping an actual count value). Document classification can be done by transforming the bag representing each document to a feature vector. These feature vectors can then be used to train classifiers. Some practical examples of this include using the naive Bayes classifier for the purpose of detecting e-mail spam[2] and training a Support Vector

Machine on bags of words representing documents[14]. This transformation of a document to a bag of words is displayed in Figure 2: a list is constructed of all possible words in the dataset, and the list (the bag of words) has non-zero values (the value 1) if the word represented by the position in the list is present in the document. This can be visualized by displaying a list of all unique words that appear in the input dataset before transformation, as done in the figure below.

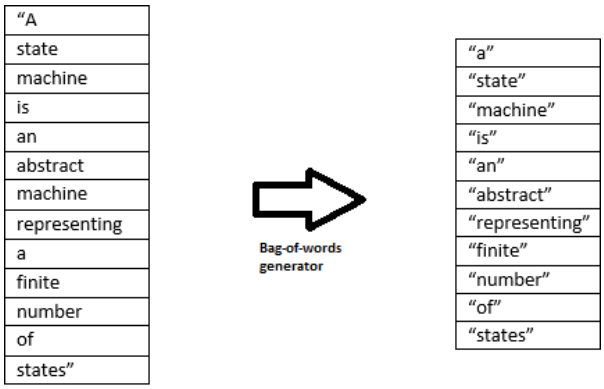


Figure 2: An example of generating a bag of words from an input sentence. The simplest form of bag-of-words generating is applied: word frequencies are not counted. In computer memory, this 'bag' would be a long list of zeroes and ones indicating whether each possible word is present in the input sentence.

Another model for document classification is **Latent Semantic Indexing**. With this method, the mathematical technique of Singular Value Decomposition(SVD) is used to reduce the size of input set objects by transforming these input objects set to objects in a latent space in which the distance between words or documents can be used to calculate similarity, as done in (Deerwester et. al., 1990)[8]. This model allows for representing documents (and invidual words) into latent values (which can that are discovered during the SVD process). An example of input documents transformed to this so-called latent space is shown in Figure 3.

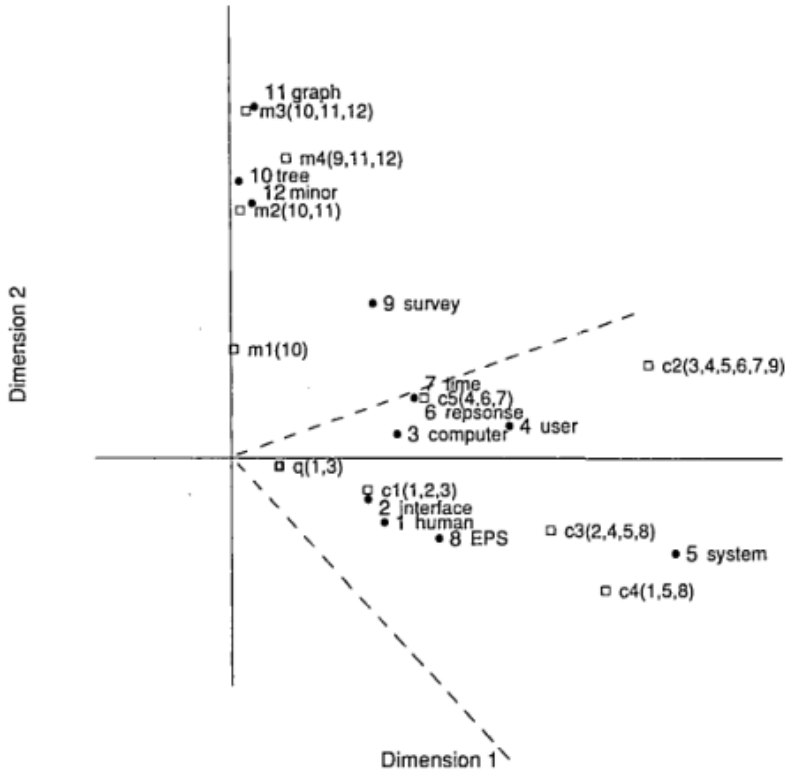


Figure 3: Example of applying SVD on an example dataset, as done in (Deerwester et al., 1990)[8], with as result two values (dimensions) for each word and document.

As seen from the latent space displayed in the figure, a way for classification would be taking the latent values of (in dimensionality reduced) objects, which are calculated by performing SVD, and using them to train and classify using a classifier, for example the Support Vector Machine(SVM)[27].

## 2.3 Neural networks and language processing

Another powerful tool for language processing is the neural network. Neural networks are data processing structures that are inspired by what is known about the working of human brains. The neural networks that are used in machine learning are also called artificial neural networks, as they attempt to replicate the way humans process information.
Neural networks consist of layers of individual neurons that, when classification is attempted using said neural network, propagate information from each layer to the next. This propagation can be done by 'connections' which form links between neurons in dif-

ferent layers. These links are represented by weights that are used to multiply neuron output values with before using them as input in neurons in further layers. Training of these neural network weights can be done by the technique of backpropagation[10]. As neural networks are extremely versatile and can be used in a variety of applications, they can also be used for in the area of document (text) processing. Several text processing problems that can be solved by artificial neural networks include, but are not limited to: machine translation[3][7], language (sentence) modelling[15][23] and named entity recognition[6][16].

## 2.4 Language processing through word embedding

Word embedding entails the use of machine learning techniques to map words to vectors containing real values. As most classifiers and machine learning techniques require (real-valued) numbers as input, word embedding enables these techniques to handle words (or sentences) as input. This principle of word embedding is nowadays very common in neural networks that tackle some kind of natural language processing problem[4][18].
Usually, trained word vectors show relations to desired output classifications or to each other. An example of this would be when the technique of word embedding is used for language modelling, i.e. predicting the next word based on a number of previous words. Table 6 displays learned word vectors within the problem of language modelling as presented in (Pennington et. al., 2014)[24]. It can be seen that, with the word embedding technique that is used, word vector (Euclidean) distance can be a good measurement for word similarity. An explanation for this would be that similar words are expected to yield similar classification outputs. This similar output will lead to embeddings being similar as well, due to the way the classifier is trained in the analysed experiment[24].
It is important to note that this example is not based on a neural network, showing that the notion of word embedding itself can be seen as a separate technique from the technique of neural networks.

Table 6: The first 10 features of 50-dimensional feature vectors of the word 'frog' and the top 7 closest words(Euclidean distance) as constructed by (Pennington, Socher & Manning, 2014)[24].

| **frog** | frogs | toad | litoria | leptodac tylidae | rana | lizard | eleuthero dactylus |
|---|---|---|---|---|---|---|---|
| 0.61038 | 0.16142 | -0.00255 | 0.47092 | 0.61412 | -0.54875 | 0.32068 | 0.80984 |
| -0.20757 | -1.0424 | -0.42187 | 0.099646 | 1.683 | -0.31127 | -0.36405 | 0.4459 |
| -0.71951 | -0.46239 | -0.92706 | -0.25081 | -1.1209 | -0.411 | -0.96793 | -1.3083 |
| 0.89304 | 0.77606 | 0.5046 | 0.97404 | 2.0704 | 0.4231 | 0.38308 | 1.0127 |
| 0.32482 | 0.26464 | -0.22752 | 0.25243 | 1.7839 | -0.25693 | 0.62911 | 1.2817 |
| 0.76564 | 0.31932 | 0.85034 | 0.31009 | -0.53265 | -0.38582 | -0.11026 | 1.0654 |
| 0.1814 | -0.76033 | 0.10351 | 1.5959 | 0.99716 | 0.742 | 0.23428 | 1.8016 |
| -0.33086 | -0.52584 | -0.40195 | -0.81645 | 0.97641 | -0.41833 | -0.72842 | 0.64034 |
| 0.79173 | 0.38692 | 0.41292 | 0.22613 | 1.8206 | 0.081392 | 0.71771 | 1.7055 |
| -0.31664 | 0.01985 | 0.13847 | -0.02093 | -0.84315 | -0.22236 | -0.3195 | 0.1681 |

## 2.5 Combining neural networks and word embedding

While standalone word embedding techniques, they can be combined with the use of neural networks as well[4][19]. As discussed earlier, neural networks consist of individual layers that are connected to each other. Word embedding can be included into a neural network simple by adding another layer to said neural network. This word embedding layer has the task of processing the input words and outputting embedding vectors coupled to said input words. Training of a word embedding layer can be done through the widely-used technique backpropogation, used in training complete neural networks[10]. When this technique of backpropagation is used, the embedding output values per word are simply changed based on the feedback from succeeding layers in the neural network. This change, or learning of the embeddings, starts with the embeddings being initialized to random values, and then being learned by this backpropagation.
Because the individual neurons in a neural network operate on real numbers, the technique of word embedding layers in a neural network is often applied when a transformation is required between the textual data found in datasets and the real numbers required by each of the layers of a neural network.

## 2.6 The attention mechanism in neural networks

In neural networks that perform image processing, attention is a way to focus on a specific part of an image. This specific part is dynamically chosen based on the input image and the trained neural network[29]. The advantage of using this attention mechanism is that there can be a focus on a specific, important part of the image, meaning less computing power is wasted on the other, less important, parts of the image. On top of this detailed focus lies another advantage: choosing the right part to focus on, the part that represents the image the best, can make sure that the clas-

sification output is the best for the whole image. This neural attention mechanism is loosely based on how the human visual attention mechanism works, with the same characteristics and advantages[30].

The attention mechanism can of course be seen as a general method for improving neural network performance, and thus it can be used on other problems as well. This mechanism has in fact been used to aid in performing machine translation[3] as well as in text comprehension and question answering[11], as shown in Figure 4.

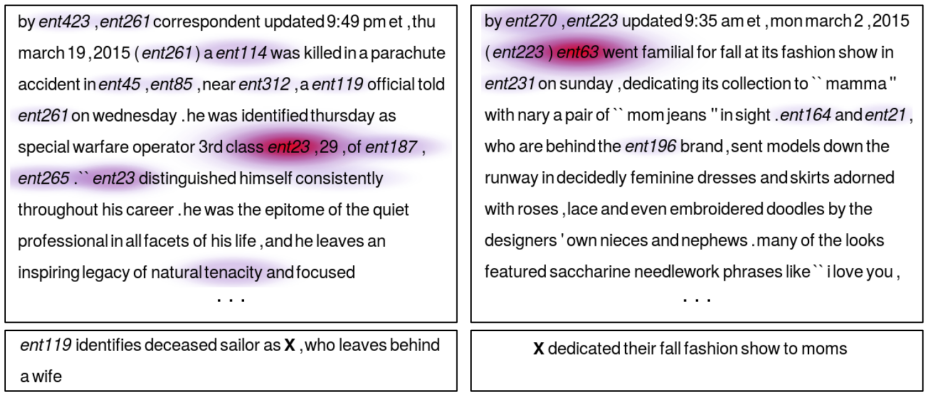


Figure 4: An example of the attention mechanism, where the important part of the input text(dependent on the query below) is correctly identified, as shown in (Hermann et al., 2015)[11].

In combination with the earlier presented concept of word embeddings, the attention mechanism can be found back as well in some instances. In these instances, the more important words in certain input texts have relatively strong (relatively high or low values) or otherwise discerning embeddings. In this case, it is important to keep in mind that this attention to certain words or parts of sentences is not always explicitly defined as a seperate value, but can be implicitly included within the learned values of a classifier. The fact that this attention is often included in a classifier in some way is also the reason it is not discussed further, but can be determined by object representation values or by learned classifier parameters, as the later learned word embeddings will show.

## 2.7 Imbalanced datasets and sampling

An often appearing problem when one wants to use a dataset to train a classifier is that this dataset contains classes or categories of objects that are not equal in size.

When this happens, it is said that the dataset is imbalanced. The main disadvantage of these imbalanced datasets is that the class or category that is the smallest is likely to have a much larger classification error of its objects, as the training of the classifier will focus more on correctly classifying the class of which there are a relatively large amount of objects. This because correctly classifying the larger amount of objects in the majority class usually results in smaller training error rates. A way of solving this problem of imbalance between classes or categories is by applying sampling. There are a multiple of sampling techniques, ranging from simply copying or removing objects from a dataset to more complex techniques in which artificial objects are added to the dataset[5][9]. In these more complex techniques, the objects that are artificially created and added to the dataset depend on the objects that are present in the imbalanced dataset.

The techniques that are used to solve the problem of imbalanced dataset in this research can be categorized in two categories: oversampling and undersampling. Several oversampling and undersampling techniques are presented, with a description on how to transform the dataset such that it is no longer imbalanced. In all of these methods, it is important to know which class or category constitutes the majority class and which class constitutes the minority class. This is usually determined by the number of objects within each of the classes.

### 2.7.1 Undersampling of the majority type

The most common and straightforward way of undersampling is simply removing random objects from the majority class or category. Using this method, objects from the majority class or category are picked at random and are removed from the dataset. This will go on until the number of majority class objects equals the number of minority class objects, thus resulting in a balanced dataset.

### 2.7.2 Oversampling of the minority type

The opposite of undersampling the majority type is the method of oversampling the minority type. Using this method, objects from the minority class are copied to increase the number of minority objects. This copying will go on until both classes are equal in size. Determining which objects to copy can be done by copying one random objects at a time or by copying the objects fairly, thus resulting in each minority objects being copied the same number of times.

### 2.7.3 Focused oversampling

Focused oversampling (sometimes called resampling or selective oversampling) consists of oversampling only the objects of the minority type that lie on the boundary between the minority and majority types. In practice, this usually results in only the minority objects that are misclassified being oversampled, instead of oversampling all the minority objects, as explained in the previous method description.

To figure out which objects are misclassified, a classifier is first trained on the original, imbalanced dataset. After that, training objects are classified to determine which of the minority class objects are being misclassified. Knowing which objects are misclassified, the focused oversampling is performed, oversampling only the objects that are classified incorrectly. This oversampling leads to an altering dataset, which is then used to again train a classifier and to again find out which minority objects are incorrectly classified.
These steps of oversampling and retraining form the core of this method and can also be described as follows:

1. If the minority class size is larger or equal to the majority class size, or if the minority class size is classified without any error, stop the method
2. Train a classifier on the dataset
3. Determine which minority objects are misclassified
4. Copy these misclassified minority objects in the dataset

In the end, this method will result in either a balanced dataset or and imbalanced dataset in which the minority objects are all correctly classified during training.

# 3 Experimental setup

As mentioned in the introduction, ths section will describe how all experiments to perform the research objectives is performed. First, a description is made of the excact classification technique that will be used, which is the word embedding neural network. Next, more is written on the dataset that will be used for the experiments, including a description on the class balance, the size of the dataset and the presence of objects relevant to the problem of negation recognition. After this information on the dataset is presented, experiments to figure out the highest performance neural network design are described. More experiments are then done on the different ways of preprocessing the dataset and whether this is a useful thing to do. Not really an experiment but still an interesting task leading to some results is the inspection of the learning word representations, which is presented in its own subsection as well. Last, ways are described to combine the classifications that are done on object subsets ($n$-grams), which will in turn form complete object classifications which can later be compared to the existing state-of-the-art.

## 3.1 Classification method: Word Embedding Neural Network

The method of classification chosen for the following experiments is a neural network. More specifically, a neural network that performs word embedding on its input, with as output a value denoting the predicted class, ranging from 0 to 1 (continuous). The reason for choosing to use a neural network is that it fits the aim of this research, which is to perform negation recognition and handling without any assumptions on the underlying data. Furthermore, it is known that a neural network is able to solve the XOR problem, which is paramount to solving the negation recognition problem. The specific reason for using a word embedding layer within this neural network is that word embedding is known to work well with machine learning problems that are based on textual data, for example the earlier mentioned research by (Bengio et. al, 2003)[4] or the more recent research by (Mikolov et. al, 2013)[19].

The design of the neural network is made with two principles in mind: achieve the highest classification performance for negation recognition (on the whole dataset) while keeping the design as simple as possible. The final design, which adheres to these two principles, is show in Figure 5. This design is not made up beforehand but is the result of the results of experiments on different design parameters. This testing of different design parameters is described in a later section.

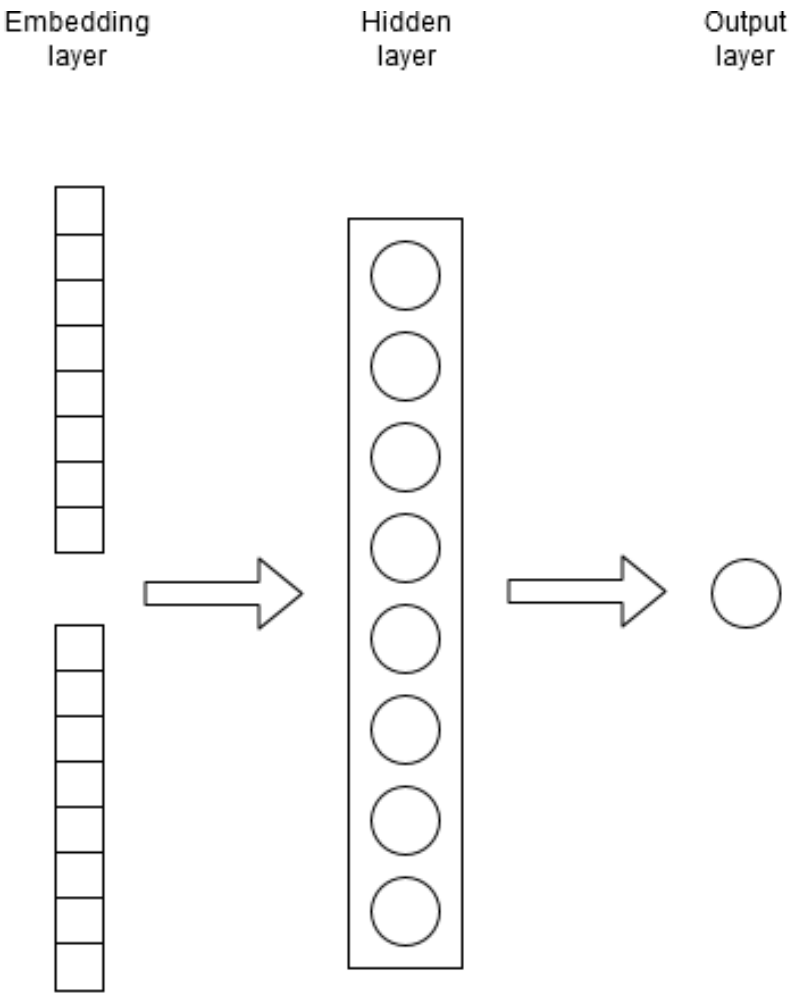


Figure 5: The design of the final neural network that is used for negation recognition and handling, with its parameters determined by experiments and presented in Table 14.

## 3.2 Dataset

Before any experiments can be done, one or multiple datasets have to be found and chosen to be used first. The dataset that is used during the experiments presented in this research is actually a combination of several distinct sentiment classification datasets[20]. These individual datasets originate from a classification competition[22] as well as from a pattern recognition laboratory at a machine learning company[1]. All of these datasets contain data from Twitter, that is to say, tweets. These tweets

are taken without any filter in mind: all information except for the tweet text and the classification is removed.
Some characteristics of this whole dataset are displayed in Table 7.

Table 7: Characteristics of the used dataset

| | |
|---|---|
| **Total number of tweets** | 1577278 |
| **Average tweet length (in words)** | 12.67 |
| **Number of unique words** | 267613 |
| **Ratio positive/negative tweets** | 50.1/49.9 |

### 3.2.1 Negation recognition objects within the dataset

As this dataset contains a large number of tweets taken without any (known and relevant) filtering criteria, it also contains a lot of words and tweets that are irrelevant for the problem of negation recognition. To determine the most relevant tweets or parts of tweets, a small list of words is made. This list can be used as an overview of some of the most appearing positive and negative words. These words, relevant to the problem of negation recognition, are listed in Table 8.

Table 8: A list of some of the most appearing positive and negative words in the complete tweet dataset

| | **Term** | **Number of appearances** |
|---|---|---|
| **Positive** | | |
| | great | 33378 |
| | nice | 23502 |
| | happy | 26358 |
| | awesome | 18124 |
| | cool | 14344 |
| **Negative** | | |
| | sad | 29229 |
| | bad | 27059 |
| | sorry | 26043 |
| | sick | 15787 |
| | boring | 4416 |

## 3.3 Neural network design

Even though the general structure of the neural network has been determined: it will be a word embedding neural network, there are still some design parameters to determine. These design parameters include the embedding dimension, as well as the structure of the hidden layers within the neural network. A detailed list of the parameters that are tested, including their possible values, is presented in Table 9.

Table 9: Description of the different neural network design parameters to test, and their tested values

| **Parameter description** | **Tested values** |
|---|---|
| Embedding dimension | 1, 2, 4, 8, 16, 32 |
| Amount of hidden layers | 1, 2 |
| Number of nodes in the first hidden layer | 1, 2, 4, 8, 16 |
| Number of nodes in the second hidden layer (if applicable) | 1, 2, 4, 8 |

## 3.4 Negation recognition and handling

Two broad strategies can be considered when attempting to solve the problem of recognizing and handling negations within text. These strategies are different in the amount of data that is used to train the classifier that will recognize and handle said negations. One option is to slim down the dataset such that only objects relevant to the problem of negation recognition are included and used during training. Another option is to decide to use the complete dataset, without removing or filtering out any of the objects. This will result in a more realistic dataset that also contains objects that are not directly relevant to the problem of negation recognition. Keep in mind that the negation recognition experiments described in this section are done on 2-grams, which are taken from the whole tweets in the dataset.

### 3.4.1 Training on a slimmed down dataset

One of the ways to train the negation classifier is to slim down the dataset before training the classifier. With this technique, preparing the dataset for training entails only including objects useful for learning to perform negation recognition and handling. The way the dataset objects are filtered is done by performing the following steps:

1. Identify the objects relevant for negation recognition
2. Filter out all objects that are considering irrelevant
3. Perform preprocessing and training of the classifier on the resulting dataset

The first step of identifying which objects are relevant is based on the list of most appearing positive and negative words, as listed in Table 8. The objects that are considered relevant for training can be described as follows: every 2-gram that contains one of the top words as listed in Table 8 as second word is considered relevant and thus included in the slimmed down dataset. Table 10 provides a good indication of the rigorousness of this filtering step. It can be seen that less than 1% of the objects are retained after filtering.

Table 10: Quantification of the filtering step

| | |
|---|---|
| **Original dataset size** | 16850698 |
| **Absolute number of objects after filtering** | 159975 |
| **Relative number of objects after filtering** | 0.95% |

The remaining dataset, after completing this process of slimming down, is still an extremely unbalanced dataset: only 2286 objects, which equals to 1.4% of the objects after filtering, contain a negation. To balance out objects of these different types, a balancing method has to be used. To find out which balancing method performs the best, experiments are done with the following balancing methods:

- Undersampling of the majority type;
- Oversampling of the minority type.

The workings of these balancing methods are described in the earlier section on the background of these methods.

### 3.4.2 Training on the full dataset

The other strategy for training the negation handling classifier is to train a classifier on the complete dataset. This way, the training is focused on more than just the relative small amount of objects that are relevant to the negation recognition problem. Furthermore, by training on the complete dataset, the neural network can learn the sentiment of more different words, including rare words or words that are not strongly positive or negative. By learning the sentiment of all of these individual words while still training the neural network to perform negation recognition and handling, a model capable of true generalization over all positive and negative words could be constructed.

When training on the complete dataset, no changes to the dataset of 2-grams should be made. This dataset, which contains all 2-grams that originate from the original set of tweets, is called the raw dataset. In the optimal situation, a classifier trained on this raw dataset should be able to perform the task of negation recognition and handling without any mistakes or errors. When classifiers trained on datasets that are alterations of this raw dataset are measured, this raw dataset will serve as a baseline and thus as a benchmark to compare these different alterations of the dataset to.

As mentioned, some alterations of the original 2-gram dataset are used to train and test the neural network as well. One of these alterations is the dataset in which negation recognition objects are artificially added to the original 2-gram dataset. Experiments are done to find out whether it would be possible to completely eliminate any misclassifications of the relevant negation recognition objects. Also, finding out whether the neural network is able to generalize over negations and relevant negation recognition objects when some of these negation recognition objects are not explicitly added (included in the artificial dataset) is a great way of testing the generalization ability of the neural network.

## 3.5 Word embeddings

Another interesting 'experiment' is looking at the word embeddings that are a result of the training of the neural network. These word embeddings can show relations between the words in an input sentence and its label. Word embeddings can also show relations between individual words and their meanings. As different neural network designs can lead to different word embeddings (with different embedding dimensions), multiple experiments can be done with these word embeddings.
Experiments on word embeddings include looking at how positive and negative words are embedding after training the neural network. This will show the possible relations between the meanings of these words and their embeddings.

## 3.6 Combining partial classifications

To find out whether the used classification technique (word embedding neural networks) is able to hold its own, not only on 2-grams but on complete short texts (in this case tweets), experiments are done to figure out the best way to combine these partial classifications. Since these experiments will result in performance figures over entire tweets, there is a baseline to compare against. According to the original research that presented the dataset that is used[20], their state-of-the-art performance is a binary accuracy of 75%. That is, 75% of test objects during their testing were correctly classified.
Since the 2-gram classification is a way of classifying a fixed length sentence, it cannot be used for real-world tweet classification. Several categories of methods for combining fixed-length classification results are:

- Taking a (weighted) average of the subset classification results;
- Using machine learning techniques with internal memory (for example recurrent neural networks) to combine subset classifications;
- Combining results by discarding certain subset classification (throwing away subset classification that are deemed 'not important').

Experiments are done on a multiple of combining methods for (partial) fixed-length classifications. These combining methods are described in the following subsections.

### 3.6.1 Combining method: Averaging subset $n$-gram classifications

The most straightforward way of combining these partial classification is by calculating the average (mean) over all $n$-gram (subset) classifications. It is important to note that, with our current practical setup of classifying 2-grams, this method will not be completely fair with respect to the first and last word in the input sentence, as these words are only present in one 2-gram, while the other words will be present in two 2-grams.

### 3.6.2 Combining method: Averaging the minimum and maximum subset classifications

Another method of combining partial classification results is to calculate the average (mean) of just the lowest and highest subset classification results. This way, the focus lies purely on the outliers (the highest and lowest subset results), meaning the other subset results are simply deemed 'not important' and are thus discarded.

### 3.6.3 Combining method: Taking the strongest of the minimum and maximum values

This combining method is similar to the previous method, in the sense that the minimum and maximum classification results are the only determinants of the full sentence classification output. However, the difference here lies in the computation of this full sentence classification output. This method simply outputs the minimum or maximum value, depending on which one is stronger. In this case, the strong value is the value that lies further away from the middle of the range (which is 0.5 as the $[0, 1]$ range is used for the output values). By applying this method, the assumption is made that this strongest value is the most important value of all the classification values. Thus, the words that result in this strongest subset classification are the most important words in the entire text.

### 3.6.4 Combining method: Taking the (first) negation subset classification

This combining method differs from the other combining methods in the sense that it can only be applied when the input sentence (and thus the corresponding subset classifications) contain a negation. In this combining method, only the classification value of the first subset that contains a negation is considered relevant and is thus taken as the result of this combining method. Because only one subset classification result is taken, it means that all other subsets are considered irrelevant and thus will have no influence on the final full sentence output.

### 3.6.5 Experimental dataset setups

As some of these combining methods can only be used when classifying sentences that are known to contain a negation, multiple experimental setups are presented. These setups differ in the dataset that will be used.
The first experimental setup consists of experiments on all tweets in the dataset. During these first experiments, the first three combining methods 'averaging subset $n$-gram classifications', 'averaging the minimum and maximum subset classifications' and 'taking the strongest of the minimum and maximum values' are tested.
The second experimental setup consists of experiments that are only done on the tweets that contain a negation. In this setup, all of the combining methods are tested and compared to each other. This results of these experiments are used not only to find out which combining method works the best in this setup, but also to draw conclusions on these special subsection of tweets that contain a negation, and their ease of

classification.

These combining methods rely on an underlying classifier to classify the subsets of entire sentences. During all of the described experiments, this classifier is the best-performed word embedding neural network as determined by the experiments on neural network design. This best design is taken and trained on the full 2-gram dataset, with a training round of 100 epochs. The word embeddings and neural network weights that result from this training are used as the classifier for the 2-grams that are later combining used the methods described in this section.

# 4 Results

This section presents results on the experiments presented in the section on the experimental setup. Each individual section shortly described the experiments again, describes the achieved results and draws a certain conclusion based on the description of the setup of the experiment and the research objectives as presented in the introduction section.
First, the results of the experiments on the neural network design are presented, this to determine the actual neural network (design) to use in the later experiments. Afterwards, experiments are done to figure out whether it would be useful to perform dataset preprocessing by slimming down the dataset to include less objects irrelevant to the problem of negation recognition. Afterwards, more dataset preprocessing results are presented, this time on the full dataset, and to the full dataset with artificial objects added. Results on the training of word embeddings is presented as well, showing the actual word embeddings and their relations between the words that are represented. Last, the results on the method for combining $n$-gram classifications are described, showing the classification performance on entire tweets.

## 4.1 Neural network design

As described earlier, it is important to know the best possible design parameters to make sure the neural network achieves optimal performance on this problem. As described in Table 9, several design parameters are tested. This testing is done by constructing a neural network with these parameters and then training and testing this neural network on the complete dataset. For training, 90% of the dataset will be used, leaving a testing set consisting of 10% of the dataset. Average errors will be taken over 5 training and testing rounds, with 20 epochs per training round. The training and testing results with these different design parameters is displayed in Table 11. Results on the classification of negation recognition objects (these objects are displayed in Table 16) are also displayed. These results are again displayed as the average Mean Squared Error (MSE) over 5 training and testing rounds.

Table 11: Different neural network design parameters and the average Mean Squared Error (MSE) over 5 training and testing rounds (with 20 epochs per training round). Sample standard deviations are displayed as well.

| **Embedding dimension** | **# of nodes in the first hidden layer** | **# of nodes in the second hidden layer** | **Test set MSE (standard deviation)** | **Negation recognition objects MSE (std. dev.)** |
|---|---|---|---|---|
| 1 | 8 | N/A | 0.227(0.000) | 0.304(0.005) |
| 2 | 8 | N/A | 0.226(0.001) | 0.215(0.047) |
| 4 | 8 | N/A | 0.225(0.000) | 0.168(0.024) |
| 8 | 8 | N/A | 0.225(0.000) | 0.124(0.018) |

| | | | | |
|---|---|---|---|---|
| 16 | 8 | N/A | 0.225(0.001) | 0.133(0.015) |
| 32 | 8 | N/A | 0.225(0.000) | 0.130(0.010) |
| 8 | 1 | N/A | 0.227(0.000) | 0.286(0.006) |
| 8 | 2 | N/A | 0.226(0.000) | 0.199(0.026) |
| 8 | 4 | N/A | 0.225(0.000) | 0.155(0.016) |
| 8 | 8 | N/A | 0.225(0.000) | 0.124(0.018) |
| 8 | 16 | N/A | 0.225(0.001) | 0.141(0.022) |
| 8 | 8 | 1 | 0.226(0.001) | 0.158(0.016) |
| 8 | 8 | 2 | 0.226(0.000) | 0.163(0.024) |
| 8 | 8 | 4 | 0.226(0.000) | 0.143(0.030) |
| 8 | 8 | 8 | 0.226(0.000) | 0.156(0.039) |

One of the results in Table 11 is the Mean Squared Error (MSE) on a test set that is taken from the dataset. What can be seen is that changing the design of the network has little to no influence on this MSE performance measurement. Modifying design parameters to make the neural network simpler or more complex does not result in a significant (if any) improvement in test performance. This performance issue could be linked to several causes, which were all investigated. The possible causes for this lack of improvement or decrease in performance with different design hyperparameters are: wrong or irrelevant error measurements, wrongly chosen other learning parameters (learning rate, number of epochs) or the word embeddings needing to be relatively simple (meaning a more complex model will not lead to improved performance). Each of these potential causes is described in the following subsections.

**Wrong or irrelevant error measurements**

One probably cause of this lack of improvement could be that the error measurement that is used, which is the Mean Squared Error (MSE), is not capturing classification improvements as well as expected (when design hyperparameters are changed). To verify whether this could be an actual reason for the performance issue, other performance measurement methods have been tried as well. These other performance measurements include absolute (non-squared) classifcation error, the Area Under Curve (AUC) error measurement and the error during training (after each epoch). Classification results measured in the absolute (non-squared) error, as well the the Area Under Curve (AUC) errors are displayed in Table 12. This table shows only minimal classifications improvements when the design complexity of the neural network is increased, with these displayed error measurements. This would mean that even when different error measurements are used to compare the networks to each other, there is still no significant improvement in performance when the neural network design increases in size and complexity. It can thus be concluded that using wrong or irrelevant error measurements it not the reason for the perceived lack of improvement when the neural network is designed to allow for more complexity.

**Wrongly chosen other learning parameters**

Besides the testing design parameters, some learning parameters also play a role in the learning and thus the performance of the training neural network. The two learning parameters that might influence the test results are the number of epochs per training

Table 12: Classification results when using error measurements other than MSE. Average results and sample standard deviations are taken over five training and testing rounds.

| **Neural network design (embedding dimension:nodes in hidden layer one)** | **Absolute average test error (standard deviation)** | **AUC error (standard deviation)** |
|---|---|---|
| 1:8 | 0.450 (0.000) | 0.672 (0.000) |
| 2:8 | 0.447 (0.002) | 0.676 (0.002) |
| 4:8 | 0.443 (0.001) | 0.681 (0.001) |
| 8:8 | 0.439 (0.002) | 0.687 (0.002) |

round and the gradient descent learning rate. As seen in Table **??**, the test performance of the classification does not significantly improve after increasing the number of epochs per training round beyond a certain amount. To confirm this, another run has been done in which the classifier is trained for 100 epochs per training round, the results of this training do not show a noteworthy improvement compared to the earlier used value of 20 epochs per training round. Thus, the amount of epochs per training round is ruled out to be a possible cause of the encountered performance issue.
Another important parameter is the learning rate parameter within the gradient descent optimization technique that is used for training the neural network. To figure out whether changing the learning rate will solve the problem of the classification performance not improving, different learning rate parameters are set as hyperparameter in neural networks that are trained and testing. The learning rate parameter values for which experiments are done are: $0.1, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0$. Experiments done with these different learning parameters did not show any improvement in performance, ruling out the choice of the learning rate parameter as a potential origin of this problem.

**Word embeddings**

Another possible source of this problem could be that the word embeddings of each input word are only needed to be relatively simple. This would mean that the neural network will train the word embeddings to be relatively simple and that the word embeddings would be comparable to each other even when the network itself becomes more complex. When these word embeddings stay relatively simple and do not contain much more hidden information, it would mean that the rest of the neural network is expected to also be relatively simple, resulting in no performance gain when the neural network is design to allow for a more complex structure.
A way to check this is to compare simple word embeddings (with a low dimensionality/vector size) to word embeddings that allow for more complex representations (having a higher embedding dimension). This comparison is done by first training the neural network with different embedding dimension hyperparameters (thus leading to different word embedding dimensions) and then performing a dimensionality reduction on these word vector of different sizes. This dimensionality reduction will result in (reduced) word vectors of the same size, which can then be compared to each other.

This dimensionality reduction is done by performing Principal Component Analysis (PCA)[28] and reducing the target embedding dimensionality to 1.
Table 13 shows the differences of the embeddings with different dimensions after the reduction step to reduce the dimensionality to 1. Also shown is the amount of variances captured in the first PCA dimension compared to a second dimension (if target dimensionality was 2 instead of 1).
The "Difference between 1D embedding and PCA result" value is calculated using the following formula:

$$avg(|w_{1d} - w_{pca\_1d}|)$$

with $w_{1d}$ the trained 1-dimensional embedding and $w_{pca\_1d}$ the $n$-dimensional embedding reduced the a 1D embedding (through PCA). This average (mean) is taken over all words for which an embedding exists (meaning all unique words in the dataset).

Table 13: Average differences of word embeddings between PCA result (reduced to 1 dimension) and the learned 1D representation by the neural network. Also shown is the percentage of variance captured in the 1st dimension, if PCA reduction to 2 dimensions is done

| **PCA performed on** | **Difference between 1D embedding and PCA result** | **Variance captured in 1st dimension compared to sum of 1st and 2nd dimension** |
|---|---|---|
| 1D representation | 0.000 | N/A (cannot reduce 1D word vector to 2D word vector) |
| 2D representation | 0.012 | 90.4% |
| 4D representation | 0.030 | 83.9% |
| 8D representation | 0.033 | 84.4% |
| Standard deviation of 1D word embeddings | 0.126 | |

When the differences between the 1D word embeddings and the results of the PCA reductions are compared to the standard deviation of the 1D word embeddings, it can be seen that these differences are (and remain) relatively small. This would mean that, even if the embedding dimension hyperparameter increases, the actual information stored and the underlying complexity of the embeddings remains almost the same. This can be further confirmed by the variances captured by the first dimension of the PCA result. Compared to the second hidden dimension (when reduction to 2D space is performed), the first hidden dimension captures a large part of the variance.
With these results in mind, it can be concluded that the issue of the performance not

increasing with a more complex neural network is caused by the fact that increasing the embedding complexity (and overall network complexity) will not result in more complex embeddings or more information (useful for the general-purpose sentiment classification) stored in the embeddings or in the rest of the neural network.

### 4.1.1 Embedding dimension

As seen in Table 11, the averaged Mean Squared Errors (MSEs) of the different embedding dimension parameters are not differing much. This means that it is necessary to take into account the Mean Squared Errors (MSEs) on the tested negation recognition objects as well. The classification errors of these negation recognition objects shown a much larger difference between the different embedding dimension parameters. As seen, having an embedding dimension of 8 will result in the least averaged MSE for the negation recognition objects.

### 4.1.2 Nodes in the first hidden layer

As the best embedding dimension parameter has been determined in the previous step, experiments can now be done to find out the best amount of nodes in the first hidden layer. Again, the averaged Mean Squared Errors (MSEs) of the different amount of nodes in the first hidden layer are not differing much. So again, the MSEs on the tested negation recognition objects have to be taken into account as well. As seen, having 8 nodes in the first hidden layer will result in the least averaged MSE for the negation recognition objects.

### 4.1.3 Second hidden layer

With both the best embedding dimension value and the best amount of nodes in the first hidden layer known, it might be useful to add another hidden layer before the final (output) layer. Measurements have been done in the exact same way as with the previous design parameters and have been displayed again in Table 11. As seen, adding a second hidden layer does not result in an improved performance compared to using only one hidden layer. Thus, a second hidden layer will not be included in the final neural network.

### 4.1.4 Final design

The parameters chosen for the final word embedding neural network are displayed in Table 14. These parameters have been chosen by inspecting the results for general sentiment classification and the results for negation recognition. As these results, which are displayed in Table 11, are mostly the same for the general sentiment classification problem, the actual decision is based on the results for the negation recognition problem.

Table 14: Chosen parameters for the final neural network, based on achieved results.

| Design parameter | Value |
|---|---|
| Embedding dimension (per word) | 8 |
| Amount of hidden layers | 1 |
| Nodes in the first hidden layer | 8 |

#### 4.1.5 Influence of the number of epochs on classification performance

To test out the learning speed of the neural network, several training experiments have been performed. These training experiments have been done using the chosen (best performing) neural network design on the same raw dataset of 2-grams. The difference in these experiments in the number of epochs per training round. As seen in the previous subsection, iterating through the entire dataset 20 times per training round (20 epochs) will result in a mean squared error of 0.124 on the negation recognition objects. The influence of using different epoch numbers per training round is shown in Figure 6, displaying the achieved testing mean squared error when different amounts of iterations through the dataset are performed.

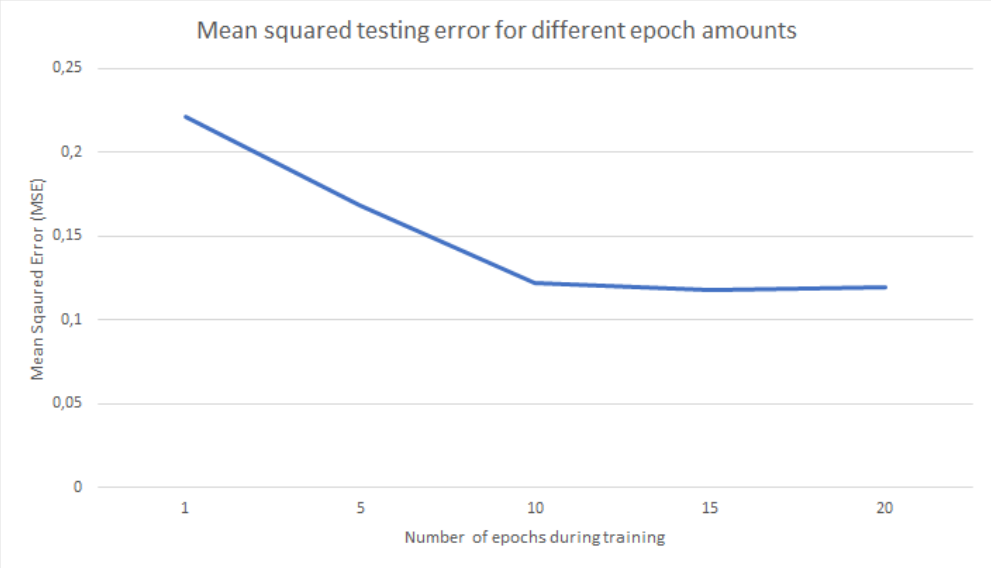


Figure 6: The influence of different epoch amounts on the achieved testing Mean Squared Error (MSE). These errors are calculated on classification averages of 5 training rounds, comparable to the process leading to the results in Table 17.

## 4.2 Slimmed down dataset

As described in the experimental setup, several ways to combat the problem of the imbalance in the training dataset can be performed. The performance of these different sampling methods are displayed in this section.
To compare each of these methods, a way of measuring the performance of these methods has to be defined first. In this case, the performance of each method is measured by calcuting the Mean Squared Error (MSE) on the negation recognition objects. These negation recognition objects are defined in Table 16, with the positive and negative terms originating from Table 8(the most important positive and negative terms in the used dataset).

Table 15: Structure and expected classification results of the objects that are used to test the trained classifiers.

| **Object structure** | **Expected classification result** |
|---|---|
| Neutral term + positive term | 1 |
| Negating term + positive term | 0 |
| Neutral term + negative term | 0 |
| Negating term + negative term | 1 |

To measure the performance of each of the sampling methods, the Mean Squared Error (MSE) is calculated per balancing method. The mean squared error works by calculating the mean of the squared differences between expected classifications of test objects and their actual classifications.
Before any of the earlier presented sampling methods are tested, a base case measurement has to be presented first. This base case, which contains the raw (unbalanced) dataset of 2-grams, will serve as a method to compare these sampling method against. The results of training and testing the neural network on the unbalanced dataset, as well as the datasets that have been balanced using the presented balancing methods, are presented in Table 16. This table includes the Mean Squared Errors (MSEs) for each of the balancing methods and for the base case.

Table 16: Results of training and testing on the datasets that resulted from different balancing methods. Classifications are averaged over 5 training rounds(with 20 epochs per training round).

| **Classified Object** | **Expected classification** | **Classification (raw dataset)** | **Classification (undersampling)** | **Classification (oversampling)** |
|---|---|---|---|---|
| 'so great' | 1 | 0.653 | 0.724 | 0.619 |
| 'not great' | 0 | 0.302 | 0.371 | 0.362 |
| 'so nice' | 1 | 0.639 | 0.603 | 0.604 |
| 'not nice' | 0 | 0.190 | 0.205 | 0.164 |
| 'so happy' | 1 | 0.809 | 0.601 | 0.790 |

| | | | | |
|---|---|---|---|---|
| 'not happy' | 0 | 0.097 | 0.126 | 0.051 |
| 'so awesome' | 1 | 0.733 | 0.719 | 0.695 |
| 'not awesome' | 0 | 0.234 | 0.350 | 0.246 |
| 'so cool' | 1 | 0.769 | 0.614 | 0.725 |
| 'not cool' | 0 | 0.145 | 0.178 | 0.133 |
| 'so sad' | 0 | 0.039 | 0.108 | 0.031 |
| 'not sad' | 1 | 0.465 | 0.110 | 0.569 |
| 'so bad' | 0 | 0.155 | 0.431 | 0.153 |
| 'not bad' | 1 | 0.614 | 0.550 | 0.656 |
| 'so sorry' | 0 | 0.093 | 0.282 | 0.093 |
| 'not sorry' | 1 | 0.361 | 0.157 | 0.422 |
| 'so sick' | 0 | 0.059 | 0.150 | 0.063 |
| 'not sick' | 1 | 0.391 | 0.109 | 0.265 |
| 'so boring' | 0 | 0.169 | 0.365 | 0.168 |
| 'not boring' | 1 | 0.533 | 0.209 | 0.683 |
| **Mean Squared Error (MSE)** | | **0.106** | **0.226** | **0.105** |

### 4.2.1 Results

With the Mean Squared Error (MSE) scores per balancing method, as seen in Table 16, the different balancing methods can be compared against each other. The relatively good performance of the base (unbalanced) dataset indicates that balancing is not necessary when the only objective is to maximize classification performance. When achieving the highest classification performance on the slimmed down dataset is not the (main) goal, it may be wise to in fact employ a balancing method. This could be done when one wants to train a classifier that is 'fair' to both types of objects (negations and non-negations). In this case, it is recommended to use the method of oversampling the minority type for balancing, as this method performed the best during testing. It is important to note that the balancing is not done on classes of objects(the classes depicting positive sentiment and negative sentiment) but rather on more complex categories of objects, which are in this case the negations and the non-negations.

## 4.3 Full dataset

### 4.3.1 Baseline performance

Before the results of the different experiments on the full dataset are listed, the classification results on the raw (baseline) dataset are presented. These classification results are presented in Table 17.

Table 17: Classification results after training on the raw (baseline) dataset. Five separate measurements are done, each with 20 epochs per training round. Sample standard deviations based on these 5 measurements are also presented. Mean Squared Error (MSE) is calculated based on the individual errors between the expected classifications and the mean actual classifications.

| **Classified Object** | **Expected classification** | **Mean Actual classification** | **Standard deviation** | **Error** |
|---|---|---|---|---|
| 'so great' | 1 | 0.752 | 0.040 | 0.248 |
| 'not great' | 0 | 0.364 | 0.056 | 0.364 |
| 'so nice' | 1 | 0.686 | 0.029 | 0.314 |
| 'not nice' | 0 | 0.324 | 0.066 | 0.324 |
| 'so happy' | 1 | 0.795 | 0.018 | 0.205 |
| 'not happy' | 0 | 0.166 | 0.035 | 0.166 |
| 'so awesome' | 1 | 0.777 | 0.040 | 0.223 |
| 'not awesome' | 0 | 0.390 | 0.035 | 0.390 |
| 'so cool' | 1 | 0.751 | 0.046 | 0.249 |
| 'not cool' | 0 | 0.225 | 0.016 | 0.225 |
| 'so sad' | 0 | 0.046 | 0.010 | 0.046 |
| 'not sad' | 1 | 0.400 | 0.113 | 0.600 |
| 'so bad' | 0 | 0.149 | 0.027 | 0.149 |
| 'not bad' | 1 | 0.574 | 0.072 | 0.426 |
| 'so sorry' | 0 | 0.118 | 0.032 | 0.118 |
| 'not sorry' | 1 | 0.449 | 0.099 | 0.551 |
| 'so sick' | 0 | 0.072 | 0.008 | 0.072 |
| 'not sick' | 1 | 0.325 | 0.040 | 0.675 |
| 'so boring' | 0 | 0.173 | 0.024 | 0.173 |
| 'not boring' | 1 | 0.520 | 0.035 | 0.480 |
| **Mean Squared Error (MSE)** | | | | **0.119** |

As seen in Table 17, results of misclassified objects are marked red. An object is considered misclassified if the rounded result value is not the same as the actual class of the object. These results mean that there is still work to be done on improving the training and construction of the actual classifier. A way of improving classification performance would be to add artificial objects to the dataset. Which objects to add is determined by the weakness of the classifier, in other words: what (type of) objects are misclassified during testing of the classifier? Results on experiments with artificially enhanced datasets are displayed in the relevant upcoming sections.

### 4.3.2 Adding artificial objects to the dataset

As seen in the previous section, using the best performing neural network design to train and test on the raw dataset of 2-grams will result in a few misclassified negation recognition objects. In this section, not the neural network design is optimized, but the input dataset itself. As seen in the previous subsection on training with a smaller and more specialized dataset, it is possible to use sampling to influence the classification results on certain objects (or the dataset as a whole). As the title of this subsection says, the method of adding additional artificial objects to the dataset is attempted. This method entails creating (or copying) relevant negation recognition objects of which the performance falls short. The aim of adding these artificial objects is to improve the negation recognition performance on these objects and preferably on the entire dataset.
Table 18 shows the Mean Squared Error (MSE) and the misclassified negation recognition objects after adding a certain amount of artificial negation recognition objects. This amount is displayed as the percentage of objects in the original raw 2-gram dataset. This original 2-gram dataset size is displayed in Table 10. The artificial objects that are added consist of randomly chosen objects from the relevant negation recognition object list as shown in Table 17.

Table 18: Classification performance after adding artificial objects. These artificial objects consist of all negation recognition objects. The number of object added is displayed as percentage of the original 2-gram dataset size.

| **# of objects added to the raw dataset** | **MSE** | **Objects misclassified** |
|---|---|---|
| 0% | 0.119 | "not sad", "not sorry", "not sick" |
| 0.1% | 0.022 | - |
| 0.5% | 0.004 | - |
| 1% | 0.001 | - |
| 2% | 0.000 | - |
| 3% | 0.000 | - |
| 4% | 0.000 | - |
| 5% | 0.000 | - |

As seen in Table 18, adding only a relatively small set of artificial negation recognition objects enable the neural network to train itself such that these negation recognition objects are no longer misclassified. The only drawback of this improvement is that additional assumptions on the dataset have to be made: the objects to include in the artificial addition set have to be chosen based on their meaning in the language of the dataset.

### 4.3.3 Generalization when adding artificial objects to the dataset

To test out whether the classifier, the neural network, is able to learn to not only correctly classify the objects that are artificially adding, but also relating negation recogntion objects, more tests are done. In this test, some of the relevant negation recognition are added, which are displayed in Table 19. The aim of this test is to figure out whether the neural network is able to generalize and improve the classifications of the other negation recognition object as well (also displayed in Table 19).
Table 20 shows that Mean Squared Error (MSE) and the misclassified negation recognition objects after adding a certain amount of these selected artificial negation recognition objects. This amount is displayed as the percentage of objects in the original raw 2-gram dataset. This original 2-gram dataset size is displayed in Table 10. The artificial objects that are added consist of randomly chosen objects from the set of relevant negation recognition objects to include (displayed in Table 19).

Table 19: Setup of the generalization experiments, presenting an overview of which objects to artificially add to the dataset and which not to add.

| **Objects artificially added** |
|---|
| 'so bad' |
| 'not bad' |
| 'so boring' |
| 'not boring' |
| **Objects not artificially added** |
| 'so great' |
| 'not great' |
| 'so nice' |
| 'not nice' |
| 'so happy' |
| 'not happy' |
| 'so awesome' |
| 'not awesome' |
| 'so cool' |
| 'not cool' |
| 'so sad' |
| 'not sad' |
| 'so sorry' |
| 'not sorry' |
| 'so sick' |
| 'not sick' |

Table 20: Classification performance after adding the selected set of artificial objects. The number of object added is displayed as percentage of the original 2-gram dataset size.

| **# of objects added to the raw dataset** | **MSE** | **Objects misclassified** |
|---|---|---|
| 0% | 0.119 | "not sad", "not sorry", "not sick" |
| 0.1% | 0.115 | 'not sad', 'not sorry', 'not sick' |
| 0.5% | 0.108 | 'not sorry', 'not sick' |
| 1% | 0.099 | 'not sorry', 'not sick' |
| 2% | 0.125 | 'not sorry', 'not sick' |
| 3% | 0.112 | 'not sorry', 'not sick' |
| 4% | 0.101 | 'not sorry', 'not sick' |
| 5% | 0.097 | 'not sorry', 'not sick' |

As seen in Table 20, adding a selected set of artificial negation recognition objects to the dataset before training the neural network will result in a slight improvement in the classification of other negation recognition objects that are not explicitly including in the artificial addition set. However, this improvement proves to be small and some of the negation recognition objects are still being misclassified. An explanation for this would be the complexity of the dataset, which often contains wrongly classified objects. Furthermore, not every 2-gram relevant to negation recognition contains the same sentiment as the entire tweet. All of this, combined with the fact that not all of these words are trained to be very similar (that is, have a similar word embedding), results in the neural network being able to only show a limited form of generalization capability.

## 4.4 Word embeddings

### 4.4.1 Embedding of sentiment

When the problem of sentiment recognition is considered, the first and foremost objective of the neural network is to learn embeddings and weights that results in positive input words being classified as a positive sentence, and vice versa. This learning of positive and negative can be seen by looking at the individual word embeddings as well. To show this, Table 21 shows the embeddings of the words that are distinctly positive or negative and are relevant to the problem of negation recognition.

Table 21: Learned 1D word embeddings of the important negation recognition terms

| **Term** | **1D embedding** |
|---|---|
| 'great' | −0.365 |
| 'nice' | −0.256 |
| 'happy' | −0.282 |
| 'awesome' | −0.362 |
| 'cool' | −0.262 |
| 'sad' | 1.62 |
| 'bad' | 0.518 |
| 'sorry' | 0.521 |
| 'sick' | 0.973 |
| 'boring' | 0.375 |

Note that it might seem strange that these positive words have such a strong negative embedding and that these negative words have such a strong positive embedding. However, this is possible due to the fact that within the whole neural network that processes these embeddings there might be multiplications with negative weights, turning these negative embeddings into positive result values for positive words and these positive values for negative embeddings into negative result values.
Another interesting set of word embeddings is the set of most positive and negative words based on their embeddings. This list of words can be computed by calculating all word embeddings through the normal neural network training and then inspecting the word embeddings that have the largest (absolute) values. For this experiment, words that appear 1000 times or less or ignored, as these words are not fairly represented in the dataset, meaning the learned sentiments values of these words might be too extreme. Table 22 lists the top 10 most positive and negative words based on their embeddings after training the word embedding neural network.

Table 22: Top 5 words that are the most positive or negative, according to their 1D embeddings

| **Term** | **1D embedding** |
|---|---|
| 'congratulations' | −0.972 |
| 'welcome' | −0.742 |
| 'www' | −0.654 |
| 'proud' | −0.646 |
| 'congrats' | −0.642 |
| 'smile' | −0.632 |
| 'thanks' | −0.569 |
| 'thank' | −0.547 |
| 'cheers' | −0.525 |
| 'hehehe' | −0.505 |
| 'sad' | 1.621 |
| 'bummed' | 1.517 |
| 'fathers' | 1.429 |
| 'gutted' | 1.400 |
| 'depressed' | 1.312 |
| 'sadly' | 1.268 |
| 'upset' | 1.187 |
| 'hurts' | 1.153 |
| 'disappointed' | 1.134 |
| 'hurting' | 1.124 |

As seen in the table, most of the words from the list of strongest embeddings are clearly (strongly) positive or negative, meaning the process of calculating 1D word embeddings is a reliable method of finding out the most positive and negative words in a sentiment dataset. Again, it can be seen that the most postive words are embedded as negative values, and the most negative words are embedded as the most positive values. This is again because the neural network will multiply these word embeddings with a negative number during the classification process, resulting in a positive output for positive sentences and a negative output for the sentences of negative sentiment.

## 4.5 Combining partial classifications

As described in the section on the experiment setups, the problem of combining partial (subset) classifications is tested used two different experimental setups. In these two experimental setups, there is a difference in the dataset but also in the combining methods that are tested. These different combining methods are displayed again in Table 23.
The results of these experiments are displayed in Table 24.

Table 23: The different combining methods, including a short description

| **Method name** | **Method description** |
|---|---|
| Averaging all $n$-grams | Taking the average of all subset $n$-gram classifications |
| Averaging min and max $n$-gram result | Taking the average of the minimum and the maximum $n$-gram classifications (out of all classification) |
| Strongest result | Taking the strongest (largest outlying) result from all $n$-gram classifications |
| Negation $n$-gram result only | Only taking the result of the negation $n$-gram |

Table 24: The classifcation errors on entire tweets (plus sample standard deviation) of the tested combining methods in the two different experimental setups. Errors and error standard deviations are taken from 5 trials.

| | **Averaging all $n$-grams** | **Averaging min and max $n$-gram result** | **Strongest result** | **Negation $n$-gram result only** | **Naive Bayes (original state-of-the-art)** |
|---|---|---|---|---|---|
| **Setup 1: all tweets** | 0.410 (0.001) | 0.403 (0.003) | 0.257 (0.006) | N/A | 0.242 (0.001) / 0.25[20] |
| **Setup 2: only tweets containing a negation** | 0.432 (0.003) | 0.426 (0.004) | 0.267 (0.006) | 0.392 (0.006) | 0.289 (0.001) |

For both the experimental setups, it can be seen that the method which takes the strongest minimum and maximum values is the best performing method for combining individual 2-gram classifcations to form a classification on a complete sentence. As mentioned earlier, the state of the art performance on these entire tweets is a binary classification performance of 75%[20], meaning this best performing combining method has a performance comparable to the state-of-the-art performance.
The fact that the other combining methods, which take more values into account and then averages them, perform less well shows that taking all subset classifications into account is a much too conservative and much too careful approach. This difference in combining method performances also show that, most of the time, only a relatively small part of the full sentence is actually important in determining whether a sentence is of positive sentiment or of negative sentiment.
Another important result is the slight underperformance of all methods on experiment setup 2: the subset of tweets that do contain a negation. A cause for this could be that these tweets that do contain a negation are often more complex and/or contain

misleading subsets ($n$-grams) of text. Often the subset of the sentence that does contain the negation is the most important subset (as seen in the performance of the method taking only the negation subset into account). However, only paying attention to the subset of the full sentence that does contain a negation will not yield the best classification results.
The last, but probably most useful result is the classificiation result of the state-of-the-art method introduced in the paper describing the original dataset[20] on the tweets containing a negation. It can be seen that using the combining method which takes the strong result into account with individual (subset) classifications originating from a word embedding neural network will result in an outperformance of the (state-of-the-art) classification method discussed by the original author of the dataset, which is the Naive Bayes classifier.

# 5 Conclusion

To conclude the research, several subsections are written to summarize what has been written in this paper. First of all, an overview of the actual experiments is given. Next, the conclusions drawn from the results of these experiments is reiterated in one section. Last, an overview of potential improvements and future work is given, highlighting the shortcomings and future challenges that arise from the drawn conclusions.

## 5.1 What has been done?

In this thesis, the problem of recognizing and handling negations has been addressed. These negations are observed in a variety of machine learning problems. In most of these problems, negation handling is a sub-problem of a much larger, more well-known machine learning problem. Apart from this theoretical notion of negations, more practical experiments with actual problems on real-world datasets have also been done. These experiments were based around the problem of sentiment classification in short texts (in this case tweets from a Twitter dataset) and the sub-problem of recognizing and handling negations to improve the overall classification of sentiment in these short texts.
With the problems of sentiment classification and negation handling within the context of sentiment classification defined, several experiments have been performed:
The first set of experiments was done to figure out the best performing design parameters of the classifier (a word embedding neural network). Several important design parameters have been identified and multiple values were tested to figure out what parameter values lead to the highest performance.
The next experiments were done on the performance of negation recognition using the neural network. Experiments have been done on 2-grams that have been taken from the tweets in the dataset. The aim of these experiments was to figure out whether it would be beneficial to perform dataset preprocessing before performing the actual classifying using the word embedding neural network. These experiments have been done on the full dataset and on a slimmed down dataset that is specifically pre-processed to only include objects relevant for negation recognition within sentiment classification.
Experiments have also been done to show the performance and ability of the word embedding layer within the neural network. Word embeddings of word relevant to the problem of negation recognition have been inspected. On the other hand, word embeddings have been inspected to figure out whether there was a relation between the values of these embeddings and the actual meanings of the words that are embedding (in the context of negation recognition for sentiment classification).
The last set of experiments was done to figure out whether it would be possible to combine said $n$-gram classifications to perform classification on whole texts. This to be able to fairly compare this combining performance to the state-of-the-art. Several combining methods have been measured, with several different dataset setups.

## 5.2 What can be concluded?

The actual conclusions from these experiments can be formulated as follows:
First of all, the formulated theoretical notion of negation recognizing and handling does exist within several larger machine learning classification problems. Some of the machine learning problems that do contain the sub-problem of negation handling include detecting sarcasm in speech classification and recognizing negations of positive or negative words in the problem of textual sentiment classification.
For negation recognition in the context of sentiment classification, using the neural network design displayed in Figure 5 results in the highest performance. Another important conclusion to draw from these design experiments is that the problem of sentiment classification itself is not complex enough to warrant a (more) complex neural network design. However, the problem of negation recognition and handling does require a more complex, less than basic, neural network design. Thus, a relatively complex neural network design has been chosen as the final design, to accommodate both of these classification problems.
The preprocessing experiments showed results that do make an argument for performing preprocessing before training the classifying neural network. However this preprocessing requires additional assumptions on the dataset: namely knowing which objects are of strong sentiment and relevant to the problem of negation recognition. In this case, when preprocessing can be performed, it is recommended to add artificial objects relevant to the problem of negation recognition to the dataset. The neural network is capable of some generalization, meaning that there is a slight increase in classification performance on some negation word structures when other negation data is explicitly added to the dataset in the form of artificial objects.
When looking at the word embeddings, the sentiment of individual words can clearly be told from their respective embeddings. This words both ways: words that are known as strongly positive or negative have relatively strong embedding values, while at the same time the strongest (both positive and negative) embedding vectors are tied to strongly positive and negative words.
Some strong $n$-gram combining methods have been found, which show a performance comparable to the state-of-the-art methods that are used. For the short texts that do contain a negation, an improvement of the state-of-the-art has been made with the training neural network combined with one of the combining methods.

## 5.3 Future work

The main shortcoming of the neural network design, as well as the experiments done is that they're only done on 2-grams taken from the entire texts to classify. Even more complex negations could be identified if the $n$-gram size would be increased. However, this would require the dataset to be of higher quality than the one used now, with more of these negations to learn from included. When a fixed input size to the neural network is required, there will always be a certain patterns in the dataset that are not recognized, as the input size is just too small to accommodate for these latent structures.

Another area in which the classifier could be improved in an improved way of combining individual subset ($n$-gram) classifications. An interesting technique for this would be to include the combining in the neural network itself. This can be done either by keeping an internal memory (for example by using a recurrent neural network) or by performing some sort of reduction of the input size (for example through convolution and pooling[13]).
Finally, can the other special situations (textual sarcasm and adverbs influencing the sentiment) within sentiment classification be solved using the same methodology and models that have been used for negation recognition handling? How will word embedding neural networks perform on these problems? Are they able to pick up/learn these complex semantic structures?
In recent machine learning research, a lot of attention has been given to neural networks for all kinds of problems. These small but often complex and challenging sub-problems, combined with the power of neural networks, could be the point of focus for future research, in which hopefully new insights are gained and performance increments are achieved.

# References


[1] S. Analytics. Twitter sentiment corpus - sanders analytics, 2011. [Online; accessed 9-November-2017].

[2] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, G. Paliouras, and C. D. Spyropoulos. An evaluation of naive bayesian anti-spam filtering. *arXiv preprint cs/0006013*, 2000.

[3] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[6] J. P. Chiu and E. Nichols. Named entity recognition with bidirectional lstm-cnns. *arXiv preprint arXiv:1511.08308*, 2015.

[7] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[8] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391, 1990.

[9] H. He, Y. Bai, E. A. Garcia, and S. Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 1322–1328. IEEE, 2008.

[10] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[11] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701, 2015.

[12] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[13] B. Hu, Z. Lu, H. Li, and Q. Chen. Convolutional neural network architectures for matching natural language sentences. In *Advances in neural information processing systems*, pages 2042–2050, 2014.

[14] T. Joachims. *Learning to classify text using support vector machines: Methods, theory and algorithms*. Kluwer Academic Publishers, 2002.

[15] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[16] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[17] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

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

[19] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[20] I. Naji. Twitter sentiment analysis training corpus (dataset), 2012. [Online; accessed 9-November-2017].

[21] T. Nitta. Solving the xor problem and the detection of symmetry using a single complex-valued neuron. *Neural Networks*, 16(8):1101–1105, 2003.

[22] U. of Michigan. Umich si650 - information retrieval - sentiment classification, 2011. [Online; accessed 9-November-2017].

[23] J. Park, X. Liu, M. J. Gales, and P. C. Woodland. Improved neural network based language modelling and adaptation. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[24] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–43, 2014.

[25] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[26] B. Sriram, D. Fuhry, E. Demir, H. Ferhatosmanoglu, and M. Demirbas. Short text classification in twitter to improve information filtering. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 841–842. ACM, 2010.

[27] J. A. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[28] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[29] T. Xiao, Y. Xu, K. Yang, J. Zhang, Y. Peng, and Z. Zhang. The application of two-level attention models in deep convolutional neural network for fine-grained image classification. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 842–850. IEEE, 2015.

[30] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015.