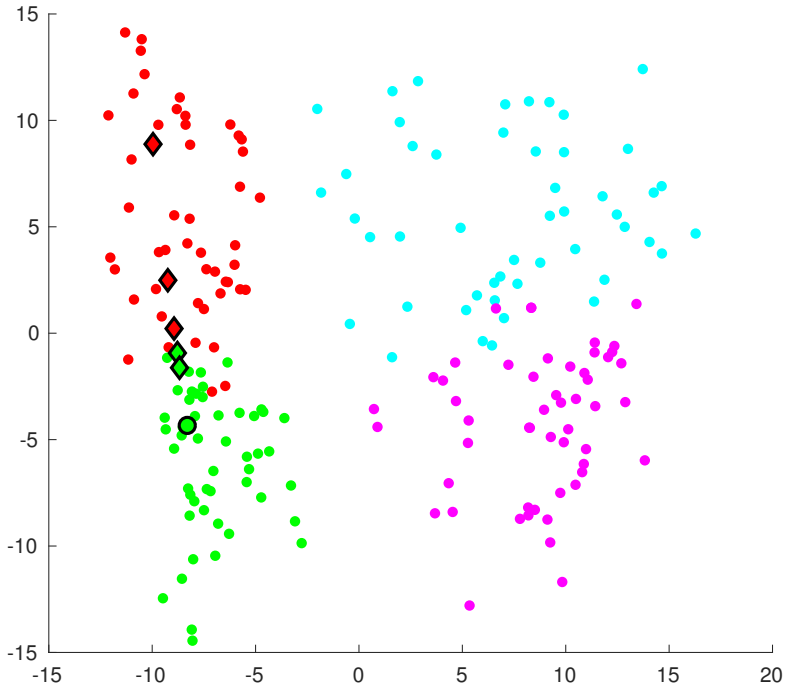
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Manifold Traversal for Reversing the Sentiment of Text

Master's thesis in Algorithms, Languages and Logic
and Complex Adaptive Systems

MARIA LARSSON
AMANDA NILSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017

Manifold Traversal for Reversing the Sentiment of Text

MARIA LARSSON
AMANDA NILSSON

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Manifold Traversal for Reversing the Sentiment of Text
MARIA LARSSON
AMANDA NILSSON

Supervisors: Mikael Kågebäck, Department of Computer Science and Engineering
Advisor: Jonatan Bengtsson, Findwise AB
Examiner: Richard Johansson, Department of Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Traversal of a feature vector, representing a sentence with negative sentiment, towards a positive sentiment. The two dimensional visualization was created using principal component analysis.

Typeset in LaTeX
Gothenburg, Sweden 2017

Manifold Traversal for Reversing the Sentiment of Text
MARIA LARSSON
AMANDA NILSSON
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Natural language processing (NLP) is a heavily researched field within machine learning, connecting linguistics to computer science and artificial intelligence. One particular problem in NLP is sentiment classification, e.g determining if a sentence holds a positive or negative opinion. There exist many established methods for solving the sentiment classification problem but none for modifying a negatively classified input so that it receives a positive classification. In this paper we propose a method for reversing the sentiment of sentences through manifold traversal. The method utilizes a convolutional neural network (CNN) and pre-trained word vectors for encoding sentences in a continuous space. The sentence representations are traversed through optimization of a test statistic as to resemble the representations of sentences with the opposite sentiment. Finally a recurrent neural network (RNN) is used for decoding the vector representation and generating new sentences.

The encoder in our model achieves 80% accuracy on the sentiment classification task and produces sentence representations in 300 dimensions. Visualizations of these representations, using PCA, shows clustering with respect to both sentiment and different topics, indicating that the representations hold information about both sentiment and textual content. Decoding the traversed feature vectors using our RNN language model produces, in most cases, understandable sentences where the sentiment has changed compared to the original sentence.

Keywords: Manifold traversal, sentiment classification, convolutional neural networks, recurrent neural networks, natural language processing

Acknowledgements

We would like to thank our supervisors Mikael Kågebäck, at Chalmers, and Jonatan Bengtsson, at Findwise, for their commitment and support throughout this project. Fredrik Axelsson, whom we also would like to thank, offered his assistance when this project was defined. Thank you all for your patience and time.

We would also like to thank Richard Johansson, our examiner, for encouraging us to develop and thoroughly motivate our project plan, at an early stage.

We thank Simon Almgren for his valuable input and help with debugging code. Sorry for the inconvenience.

Finally we would like to thank Greta.io and Google for the trip to San Francisco and the free GPU usage on Google Cloud Platform.

Maria Larsson and Amanda Nilsson, Gothenburg, June 2017

Contents

List of Figures

1
Introduction

As we live in an increasingly digitalised society, algorithms for text analysis can be used for a variety of purposes and may greatly relieve manual work. Machine learning is commonly used to predict labels for different types of data. In contrast, this project uses machine learning algorithms for changing a labelled input in such a way that it is classified with the opposite label. This is achieved by defining and traversing a manifold from the source classification to the target classification.

1.1 Background

Natural language processing (NLP) is a heavily researched field within machine learning, connecting linguistics to computer science and artificial intelligence. Some examples of NLP tasks are machine translation, part of speech tagging, question answering, text summarization and sentiment analysis. The latter, sentiment analysis, also known as opinion mining, is the task of deciding what opinion or emotion an input is holding. One example is deciding if a text is positive or negative. Sentiment analysis is used primarily for predictions and decision making. A positive post in a popular blog or a tweet from a popular person can sway the public opinion about basically anything. Consumers tend to base their purchases on the opinions of others. A lot of opinions are published on the Internet and it may be hard to find the right site and to filter among different posts and opinions. Sentiment analysis can be used to classify different comments and give an overall opinion. This is not only useful for consumer products and services, but also for political elections, health care, social events, predicting sales performance or changes in the stock market, troll filtering in social media and enhancing anti-spam filtering.

Many machine learning applications are about classifying an input. Szegedy et al. [1] showed that it is also possible to introduce small random perturbations in an input such that the predicted label changes. An interesting question that arises is whether it is possible to find a meaningful way of changing the input so that the label changes accordingly. This concept can be referred to as *manifold traversal*. A manifold is a topological space in n dimensions, e.g a plane, sphere or cylinder in 3 dimensions. The problem becomes to find an underlying manifold that may be traversed in order to change the input. This concept has not previously been applied in NLP. Take for example "I love ice cream" as input, this sentence can be classified as positive, as it says something positive about ice cream. The task is to change the sentence into something negative about ice cream.

Changing text with a negative classification so that it receives a positive classification can be useful in writing, in a similar way spell checking is used today. The way spell check is used is that misspelled words, or bad grammar, are highlighted and accompanied with a suggested change. Similarly, very negative sentences could be highlighted and a suggestion on how to change the sentence into being more positive could be presented. The ability of generating new sentences with the opposite sentiment can also be useful in data augmentation for machine learning tasks, where the amount and quality of data can be a limitation.

1.2 Project aim

The aim of this research project was to develop an algorithm for transferring a sentence with negative sentiment to a positive sentiment, and vice versa. We wanted to examine whether it was possible to find a representation of sentences that could be both encoded to, and decoded from, a continuous space.

1.3 Problem definition

The goal in this project was to develop a machine learning algorithm for changing the sentiment in a sentence. The problem was divided into three subproblems:

- representing sentences in a continuous space
- exploiting the sentence representation and traversing the manifold in such a way that the sentiment changes
- generating a new sentence from the representation space

In order to use continuous optimization methods for the manifold traversal it was desirable to work with continuous sentence representations. Since words belong to a discrete space, it was not intuitive how such a sentence representation should be achieved. It was desirable for sentence representations to be independent of sentence length and also for original and generated sentences to be independent of their respective lengths. The reason for having length independence was that an intuitive way to reverse the sentiment in a sentence is by adding or removing the word 'not'.

The manifold traversal needed to be guided by some carefully selected optimization criteria. Because only the sentiment of a sentence should change in the manifold traversal, the method was to encourage changes of the sentence representation towards the opposite sentiment whilst penalizing other changes. On the other hand, the method was not to encourage coarse changes that may cause us to deviate from, or move too far along, the manifold.

To generate a sentence from the sentence representation, arrived at through manifold traversal, an inverse transformation of the encoding procedure was needed. This inverse transformation required the sentence modelling to be invertible or differentiable. Given an output sentence, we wanted the following three points to hold for

the algorithm to be considered correct:

- The sentiment of the output sentence must be the opposite of the sentiment of the source sentence.
- The output sentence must preserve the semantic similarity of the source sentence, e.g. if the source sentence is about ice cream, then the output sentence should also be about ice cream.
- The output sentence should be grammatically correct.

1.4 Related work

In a series of experiments, performed by Kim [2], a simple convolutional neural network was trained and evaluated on different sentence classification tasks. The tasks were: binary and fine-grained sentiment classification, subjectivity analysis and question classification. The latter is the task of deciding which of 6 question types a question belongs to. From the different experiments, they were able to show that a convolutional neural network (CNN) using a single convolutional layer performed well on all tasks. Furthermore, their results showed that building the network on top of pre-trained word vectors developed by Mikolov et al. [3] significantly increased classification performance compared to using randomized word vectors.

Gardner et al. [4] developed a general purpose method for changing labels in images. Various label changing problems such as face aging, changing hair color and changing winter scenes to summer scenes were solved using deep manifold traversal. The algorithm was based on the idea that natural images incorporate a low dimensional manifold which may be traversed in order to make meaningful changes to an image. The method presented in [4] transformed images to a deep feature space using a CNN and then traversed this space towards the target features. A new image was then reconstructed from the deep feature representation.

Text prediction, or language modelling, can be performed on both word and character level. Graves [5] implemented and evaluated two language models, one word-by-word and one character-by-character long short-term memory recurrent neural network. The motivation for using character level language modelling is that the vocabulary becomes very small. A disadvantage is that recurrent neural networks (RNNs) are unable to remember past information for a long time and character level language models depend on this to a greater extent than word level language models do. Both models were tested on the Penn Treebank data set and the results suggested that the word-level RNN performed better than the character-level RNN.

Recently, Radford et al. [6] implemented a byte-level recurrent language model to generate text. The model was then trained on additional tasks such as sentiment analysis and paraphrasing. The model consisted of a single layer multiplicative long short-term memory (mLSTM) cell and when trained for sentiment analysis it achieved state of the art on the movie review data set[1] created by Pang and Lee

[1] https://www.cs.cornell.edu/people/pabo/movie-review-data/

[7] in 2005. They also found a unit in the mLSTM that directly corresponds to the sentiment of the output.

1.5 Limitations

This project was restricted to changing the sentiment of sentences containing no more than 30 words. The model has a fixed vocabulary and does not recognize other words. Instead these words are swapped for an "Unknown"-token. Similarly all digits are represented with a "Digit"-token. The smaller vocabulary resulted in less computations and hence a faster training procedure. However, the algorithm is not optimized for speed, since the aim was primarily to investigate if it was at all possible to traverse the manifold in a sentence representation space.

2
Theory

This chapter gives an introduction to artificial neural networks and detailed descriptions of *feed forward*, *convolutional* and *recurrent* neural networks. In particular, this chapter focuses on the applications of neural networks in natural language processing. Furthermore, this chapter contains the theory behind the *maximum mean discrepancy* test statistic, used in a central part of this project, and the optimization method used in combination with this statistic. Finally, details on a dimensionality reduction method, used for visualizing the results, are given.

2.1 Word embeddings

In NLP, it is often desirable to work with vector representations of words. One way is to use one-hot vectors. A one-hot vector is a vector of zeros in all but one element. For example, all rows in an identity matrix are one-hot vectors. However, this representation describes all words as equidistant and encodes no relationship between similar words. Another way to represent words is to use *word embeddings* [3, 8], which is the representation of words as dense real-valued vectors. The word embeddings are based on the distributional hypothesis [9], which states that words that are used in the same contexts tend to have similar meanings. Hence, the semantics and syntactic information about a word can be captured by assigning similar embeddings to words that occur in similar contexts. The advantage of using word embeddings, instead of one-hot vectors, in neural networks is that the network is able to generalize from previously seen data when presented with new data.

2.2 Artificial Neural Networks

Artificial neural networks (ANNs) can be applied to solve various problems such as function approximation, pattern classification, and object recognition. Inspired by the cooperation of neurons in the human brain, ANNs connect a large number of basic computation units that are adapted for solving specific tasks [10]. A single computation unit, called a neuron, is illustrated in figure 2.1. A neural network consists of many neurons put together as a network, where the output from one neuron is the input to other neurons. The output from the neuron in figure 2.1 can be calculated as

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

where x_i are the input signals, w_i are the weights that correspond to each input signal, and f is an activation function.

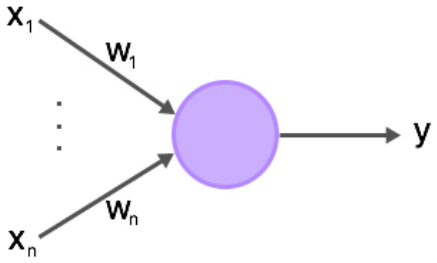


Figure 2.1: The computation unit has n input signals, $x_1, \ldots, x_n$, and one output signal, y. Each input signal has a weight, w, which regulates its significance to the output.

Neural networks are trained through presenting them with a set of training examples. Each example consists of an input and its corresponding output. For example, if a neural network is trained for sentiment classification on sentences, the training data set can contain positive and negative sentences as inputs and binary labels as outputs.

In the following sections three types of neural networks used in this project are described: feedforward neural networks, convolutional neural networks and recurrent neural networks.

2.2.1 Feed forward neural networks

The feedforward neural network (FFNN) can be seen as a weighted directed acyclic graph where the nodes are the neurons and the edges are the links between the neurons. The neurons are typically structured in different layers: first an input layer and last an output layer and an arbitrary number of so called *hidden* layers in between. A fully connected FFNN neural network is when all neurons in each layer have a connection to all neurons in the next layer. In figure 2.2 a fully connected FFNN is illustrated. The connections between neurons in different layers are weighted depending on their significance.

We denote the weights in a FFNN $w_{i,j}^{(l)}$, where i is the index of the neuron sending the signal, j the index of the neuron that is receiving the signal and l is the layer. For example, between input neuron x_2 and the hidden neuron h_3 the weight is denoted $w_{2,3}^{(1)}$. To calculate the output of the network in figure 2.2 we first calculate the values of the hidden neurons

$$\begin{aligned} h_1 &= f(x_1 w_{1,1}^{(1)} + x_2 w_{2,1}^{(1)} + x_3 w_{3,1}^{(1)}) \\ h_2 &= f(x_1 w_{1,2}^{(1)} + x_2 w_{2,2}^{(1)} + x_3 w_{3,2}^{(1)}) \\ h_3 &= f(x_1 w_{1,3}^{(1)} + x_2 w_{2,3}^{(1)} + x_3 w_{3,3}^{(1)}) \end{aligned} \tag{2.1}$$

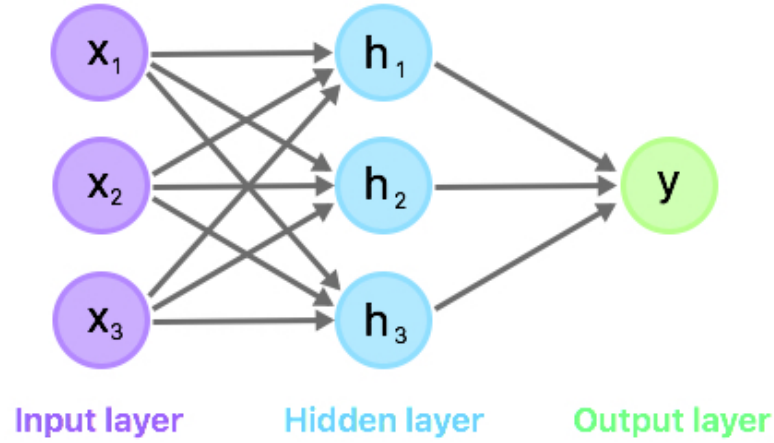


Figure 2.2: Example of a fully connected FFNN with one hidden layer. The neurons in the input layer are denoted x_1, x_2, x_3 (purple circles), the neurons in the hidden layer are denoted h_1, h_2, h_3 (blue circles) and the neuron in the output layer is denoted y (green circle). The connections between the neurons in the different layers are represented by an arrow.

and then the output,

$$y = f(h_1 w_{1,1}^{(2)} + h_2 w_{2,1}^{(2)} + h_3 w_{3,1}^{(2)}) \tag{2.2}$$

where f is an activation function.

If the inputs are denoted as a vector $\mathbf{x}$, the weights between the input layer and the hidden layer are denoted with the matrix $\mathbf{W}^{(1)}$, the weights between the hidden layer and the output layer with the matrix $\mathbf{W}^{(2)}$, then equations (2.1) and (2.2) can be written as

$$\mathbf{h} = f(\mathbf{W}^{(1)}\mathbf{x}) \tag{2.3}$$

and

$$\mathbf{y} = f(\mathbf{W}^{(2)}\mathbf{h}), \tag{2.4}$$

where f is an activation function. Three typical activation functions used in neural networks are the sigmoid function,

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

the hyperbolic tangent,

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

and the rectified linear function,

$$\mathrm{ReLu}(x) = \max(0, x). \tag{2.5}$$

2.2.2 Convolutional neural networks

Convolutional neural networks (CNNs) were originally invented for computer vision but have shown to be effective for NLP and have recently achieved remarkably strong performance on sentence classification [2, 11, 12]. A simple CNN consists of an input layer, one or more convolutional layers followed by pooling layers and a classification layer [2]. This section will explain the basics of a CNN and the structure of the described CNN is illustrated in figure 2.3.

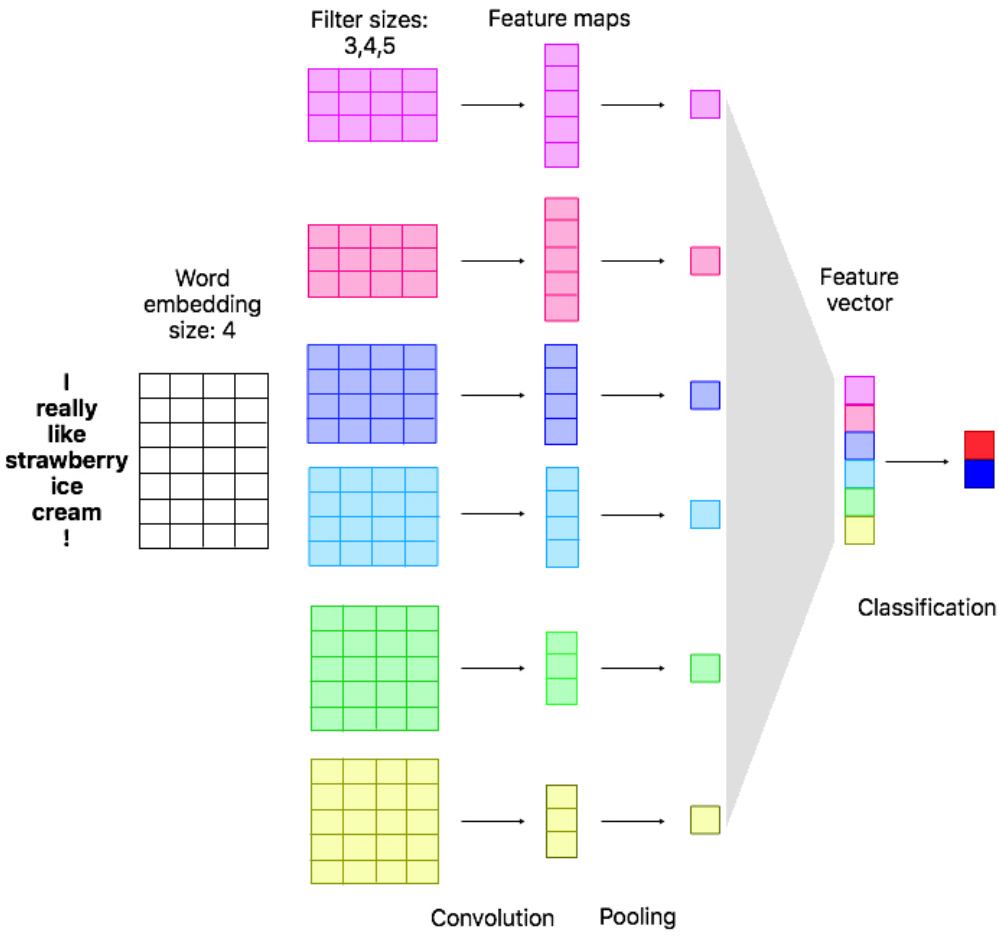


Figure 2.3: This model describes a single layer CNN. The input "I really like strawberry ice cream!" is mapped to a sentence matrix where each row corresponds to a word embedded as a vector (embedding size 4 in the illustration). In this illustration there are three filter sizes: 3, 4 and 5, and two filters of each size. Each filter is applied to the sentence matrix resulting in one feature map for each filter. In this illustration one value is extracted from each feature map in the pooling layer. The extracted values are concatenated into a feature vector which is connected to a fully connected feedforward layer. The last layer classifies the label and returns a probability distribution over the labels. In the illustration there are two labels (red and blue).

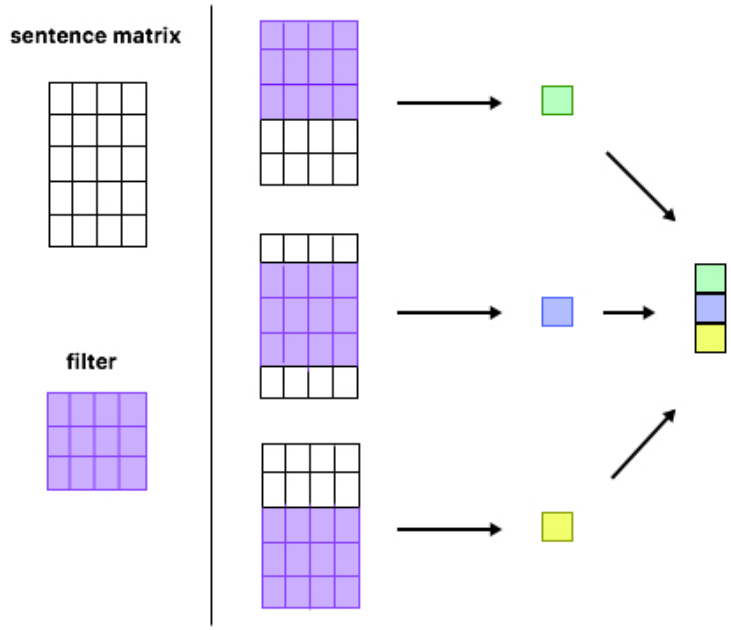


Figure 2.4: The convolution operation on a sentence matrix. The sentence matrix is illustrated in white and contains 5 rows, each representing one word. The convolutional filter is illustrated in purple and is of height 3. Hence, the filter is applied to each substring of 3 consecutive words of the sentence. From equation (2.6) the result is a scalar (green, blue and yellow). Finally the scalars from each application are concatenated into a feature map.

Given an input sentence, each word is mapped to a d-dimensional vector $\mathbf{w}_i$. For a sentence with s words, the word vectors constitute a sentence matrix

$$\mathbf{S} = \begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_s \end{bmatrix}$$

which is fed to the convolution layer. In the convolutional layer the features of the sentence are extracted. The convolution operation applies a matrix, called a *filter*, to the sentence matrix. Since each row in the sentence matrix represents one word it is common to use filters with the same width as the word representation. The height of a filter determines how many subsequent words are looked at in one application. Given a convolutional filter $\mathbf{M}$ of size $m \times d$ and the sentence matrix $\mathbf{S}$, one application of the filter results in a matrix

$$\mathbf{A} = \mathbf{M} \odot \mathbf{S}_{i:i+m-1},$$

where $\mathbf{S}_{i:i+m-1}$ is the set of m consecutive words in the sentence matrix $\mathbf{S}$ and $\odot$ is the elementwise product. The entire convolution operation results in a feature map $\mathbf{c} = [c_1, \ldots, c_{s-m+1}] \in \mathbb{R}^{s-m+1}$, where each feature, c_i is given by

$$c_i = f\left(\sum_{k,l} (a_{k,l}) + b_i\right), \tag{2.6}$$

where $b_i \in \mathbb{R}$ is a bias, f is an activation function and the summation is over all elements $a_{k,l} \in \mathbf{A}$. The operation is illustrated in figure 2.4.

The feature maps from the convolutional layer are given as input to the pooling layer. The purpose of the pooling layer is to capture the most important features of the input sentence. To accomplish this, a *pooling function* is applied to the feature maps to combine the features from the convolutional layer into one fixed size vector. A common pooling function is the max-function, which extracts the maximum value from each feature map. The extracted values are then concatenated into one vector which represents the features of the input. This vector is then passed to a fully connected layer which predicts the label.

2.2.3 Recurrent neural networks

Recurrent neural networks (RNNs) have proven to be successful in capturing the semantic composition in text [13] and are used in NLP tasks such as speech recognition [14] and machine translation [15]. Sentences may be viewed as sequences of words and given such a sequence, an RNN can be used for building a probabilistic model for predicting the next word, given the previous words in the sequence. Which words have the largest probability depends on the data set that is used for training the network. The RNN recursively processes the sequence of inputs. To make use of the sequential information, such as the semantics in a sequence of words, a state $\mathbf{s}$ is updated and passed on from each time step to the next. A basic RNN model is illustrated in figure 2.5.

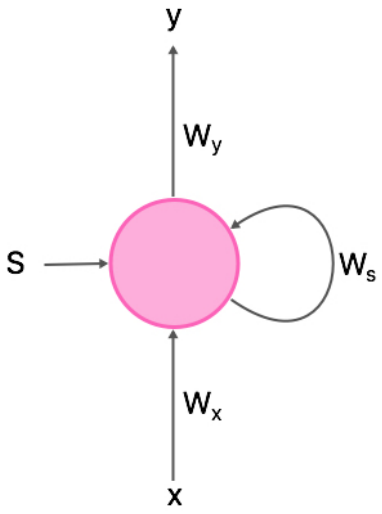


Figure 2.5: A basic RNN unit. The state $\mathbf{s}$ is calculated given the input vector $\mathbf{x}$ and weight matrices $\mathbf{W}_x$ and $\mathbf{W}_s$. The output distribution $\mathbf{y}$ is then calculated from the state and the weight matrix $\mathbf{W}_y$.

The input to the network is a sequence of words represented as one-hot vectors $\mathbf{x} \in \mathbb{R}^v$, where v is the number of words in the vocabulary known to the network. Given the input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ the state, $\mathbf{s} \in \mathbb{R}^m$, is updated as

$$\mathbf{s}_t = f(\mathbf{W}_x \mathbf{x}_t + \mathbf{W}_s \mathbf{s}_{t-1}) \tag{2.7}$$

where $\mathbf{W}_x$ is the weight matrix for the inputs, $\mathbf{W}_s$ the weight matrix for the state and f is an activation function, see figure 2.5. The output from the network is given

by

$$\mathbf{y}_t = \sigma(\mathbf{W}_y \mathbf{s}_t + \mathbf{b}),$$

where $\mathbf{W}_y$ is the weight matrix for the output, $\mathbf{b}$ is the bias and σ is the softmax function

$$\sigma(\mathbf{x})_j = \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}} \qquad \text{for } j = 1, \ldots, n.$$

2.2.3.1 Gated recurrent unit

While RNNs in theory are capable of processing information an arbitrary number of time steps back it is in practice difficult to achieve such a behaviour using gradient-descent training [16]. During training, the gradient is multiplied by the weight matrix once for every time step. If the weights are very small or large this can lead to an exponential decay or blow up of the gradient. A gradient that blows up results in oscillating weights and the network cannot be trained. Conversely, if the gradient vanishes it becomes hard to capture the long-term dependencies. These problems can be reduced through remodelling the conventional recurrent unit. One such model is the gated recurrent unit (GRU) which was proposed by [15].

With the GRU, [15] introduced a reset- and an update gate. The reset gate allows the network to forget information that is irrelevant at a later time step. The update gate promotes long term memory of the network by controlling to which extent information from a previous time step influences the current time step. Denoting the reset gate $\mathbf{r}$ and the update gate $\mathbf{u}$ the update of the hidden state $\mathbf{s}$ can be described mathematically with the following equations. The reset gate is computed as

$$\mathbf{r} = \sigma(\mathbf{W}_{x,r}\mathbf{x}_t + \mathbf{W}_{s,r}\mathbf{s}_{t-1})$$

where $\mathbf{x}_t$ is the input to the recurrent unit, $\mathbf{W}_{x,r}$ and $\mathbf{W}_{s,r}$ are weight matrices, $\mathbf{s}_{t-1}$ is the previous hidden state and σ is the softmax function. Using the same notation, the update gate is expressed as

$$\mathbf{u} = \sigma(\mathbf{W}_{x,u}\mathbf{x}_t + \mathbf{W}_{s,u}\mathbf{s}_{t-1})$$

where $\mathbf{W}_{x,u}$ and $\mathbf{W}_{s,u}$ are weight matrices. Finally, the state $\mathbf{s}$ is updated as

$$\mathbf{s}_t = (\mathbf{1} - \mathbf{u}) \odot \widetilde{\mathbf{s}} + \mathbf{u} \odot \mathbf{s}_{t-1} \tag{2.8}$$

where $\mathbf{1}$ is a vector of all ones, $\odot$ denotes elementwise multiplication and the vector $\widetilde{\mathbf{s}}$ is given by

$$\widetilde{\mathbf{s}} = \tanh(\mathbf{W}_x\mathbf{x}_t + \mathbf{W}_\mathbf{s}(\mathbf{r} \odot \mathbf{s}_{t-1})).$$

Here, $\mathbf{W}_x$ and $\mathbf{W}_\mathbf{s}$ are weight matrices. The update rule for the state $\mathbf{s}_t$ described in equation (2.8) can thus replace the rule previously used in a conventional RNN, described in equation (2.7).

2.2.4 Gradient descent

When training a neural network, the goal is to minimize an error function. Training the network requires a set of inputs $\mathbf{x}$ with known outputs $\mathbf{y}$. A common error function is the cross-entropy error function,

$$E(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i y_i \log \hat{y}_i, \tag{2.9}$$

where $\mathbf{y}$ is the target output and $\hat{\mathbf{y}}$ is the predicted output from the network. This equation implicitly depends on the weights of the network, as defined in equations (2.3) and (2.4) for the simple FFNN. The gradient descent method can be used for updating the weights. The idea is to move the weights, $w_{i,j}^{(l)} \in \mathbf{W}$, in the direction of the negative gradient $\partial E/\partial w_{i,j}^{(l)}$ such that the error function is minimized. For each training example the gradient is calculated and the weights are updated according to the gradient descent algorithm

$$w_{i,j}^{(l)} = w_{i,j}^{(l)} - \eta \frac{\partial E}{\partial w_{i,j}^{(l)}},$$

where $\eta \in \mathbb{R}$ is the learning rate. In order to update all weights in the network, the error is backpropagated from the output through the network.

When training an RNN, the current state depends on the previous states so the gradients have to be backpropagated from time t, through the network, to the first time step. Because the weight matrices are shared between time steps, the gradients have to be summed over the time steps. Thus, when training the RNN, the weights are updated as

$$w_{i,j} = w_{i,j} - \eta \sum_t \frac{\partial E(t)}{\partial w_{i,j}}, \tag{2.10}$$

where η is the learning rate and $E(t)$ is the error at time step t.

2.2.5 Preventing overfitting

To prevent neural networks from overfitting a technique called dropout can be used [17]. The idea behind dropout is to randomly disable weights in order to prevent their co-adaptation. Dropout is implemented through creating a mask of Bernoulli random numbers, equal to 1 with a probability p and otherwise 0, and apply that mask to the weights of the network. If p is close to 1 few units are dropped and vice versa. Specifically, in the CNN described above, dropout can be applied to the fully connected layer during training. In the RNN, dropout can be applied to the input of the GRU.

Another technique for preventing overfitting is to use regularisation. Regularisation introduces an additional term to the loss function used in training. In a CNN *l2*-regularisation can be implemented through adding the *l2*-norm of the weight matrices as a term in the loss function.

2.3 Maximum mean discrepancy

The maximum mean discrepancy (MMD) [18] is a test statistic used to determine whether two distributions are the same. This statistic is useful when, for example, determining whether measurements from two setups of the same experiment may be analyzed jointly. Another application is to use the statistic for distinguishing sick people from healthy people, when analyzing tissue samples [18].

Given two distributions, $\mathcal{P}_{\text{source}}$ and $\mathcal{P}_{\text{target}}$, the objective of the MMD is to find a smooth function which is large for samples from $\mathcal{P}_{\text{source}}$ and small for samples from $\mathcal{P}_{\text{target}}$. Given such a function the MMD is the difference between the mean function values for the two sets of samples. Gretton et al. [18] presents an empirical estimate of the MMD:

$$\text{MMD}(\mathcal{F}, X, Y) = \sup_{f \in \mathcal{F}} \left(\frac{1}{m} \sum_{i=1}^{m} f(x_i) - \frac{1}{n} \sum_{i=1}^{n} f(y_i) \right) \tag{2.11}$$

where $X = [x_1, x_2, \ldots, x_m]$ are samples drawn from the source distribution $\mathcal{P}_{\text{source}}$ and $Y = [y_1, y_2, \ldots, y_n]$ are samples drawn from the target distribution $\mathcal{P}_{\text{target}}$. The function f belongs to a class, $\mathcal{F}$, of smooth functions and should be chosen as to maximize the difference between the mean values of f applied to X and Y. In both [18] and [4], $\mathcal{F}$ is a reproducing kernel Hilbert space allowing comparison of multi-dimensional feature vectors. The function f^* attaining the supremum in equation (2.11) can be empirically estimated as

$$f^*(z) = \frac{1}{m} \sum_{i=1}^{m} k(x_i, z) - \frac{1}{n} \sum_{i=1}^{n} k(y_i, z), \tag{2.12}$$

where $k(x, x')$ is a kernel function. The method presented by Gardner et al. [4] uses a Gaussian kernel function

$$k(x, x') = e^{-\frac{1}{2\sigma}|x - x'|^2}$$

with σ being the kernel bandwidth.

2.4 Broyden-Fletcher-Goldfarb-Shanno algorithm

The Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS)[19] is a quasi-Newton optimization method for real-valued, multivariate, functions. Given an objective function $f(\mathbf{x})$, the algorithm finds a local minimum, $\mathbf{x}^*$. The algorithm updates an initial guess, $\mathbf{x}_0$, until a minimum is found. In the traditional gradient descent method, the current guess is updated as

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \lambda \nabla f(\mathbf{x}_n)$$

where λ is the step size and $\nabla f(\mathbf{x}_n)$ is the gradient of f in $\mathbf{x}_n$. In quasi-Newton methods, however, the gradient is multiplied by the inverse of an approximation of

the Hessian at the minimum, $\mathbf{H} \approx \nabla^2 f(\mathbf{x}^*)$ [19]. This approximation is also updated in each iteration of the algorithm. In BFGS $\mathbf{H}$ is updated as

$$\mathbf{H}_{n+1} = \mathbf{H}_n + \frac{\mathbf{y}\mathbf{y}^\top}{\mathbf{y}^\top \mathbf{s}} - \frac{(\mathbf{H}_n \mathbf{s})(\mathbf{H}_n \mathbf{s})^\top}{\mathbf{s}^\top \mathbf{H}_n \mathbf{s}}, \tag{2.13}$$

where

$$\mathbf{y} = \nabla f(\mathbf{x}_{n+1}) - \nabla f(\mathbf{x}_n) \qquad \text{and} \qquad \mathbf{s} = \mathbf{x}_{n+1} - \mathbf{x}_n.$$

The algorithm consists of the following steps:

1. Find a search direction $\mathbf{d} = -\mathbf{H}_n^{-1} \nabla f(\mathbf{x}_n)$.
2. Update the current guess $\mathbf{x}_{n+1} = \mathbf{x}_n + \lambda \mathbf{d}$, where an optimal λ is found through line search.
3. Update the approximation of the Hessian, $\mathbf{H}$, as in equation (2.13).

The above steps are iterated until a minimum $\mathbf{x}^*$ is found.

2.5 Principal component analysis

Principal component analysis (PCA) is a method used to give a simplified view of multidimensional data. PCA can be used for data reduction, outlier detection, classification, prediction etc. [20]. Given a matrix of data, where each row corresponds to an observation and each column corresponds to a variable, the goal is to project this matrix on to a subspace with fewer dimensions.

Let $\mathbf{X}$ be a data matrix with k rows and n columns. The first step in PCA is to compute the mean for all variables

$$\mu = \frac{1}{k}(\mathbf{x}_1 + \cdots + \mathbf{x}_k)$$

and re-center the data around the mean forming a new matrix, $\mathbf{B}$, which is the elementwise subtraction of μ from each row of $\mathbf{X}$. The next step is to compute the $n \times n$ covariance matrix

$$\mathbf{S} = \frac{1}{k-1} \mathbf{B}^\top \mathbf{B}.$$

Since $\mathbf{S}$ is symmetric it may also be orthogonally diagonalized. The eigenvalues, $\lambda_1, \ldots, \lambda_n$, of $\mathbf{S}$ are sorted in decreasing order and their corresponding orthonormal eigenvectors, $\mathbf{u}_1, \ldots, \mathbf{u}_n$ are the principal components for the data matrix [21]. The variance of the data is greatest in the direction of the first principal component and so on. In order to achieve a representative dimensionality reduction in, for example, two dimensions, the data matrix may be projected onto the plane spanned by $\mathbf{u}_1$ and $\mathbf{u}_2$.

3
Model

As stated in the problem definition the project consists of three subtasks. The first task is representing sentences in a continuous space. The second task is exploiting the sentence representation and traversing the manifold in such a way that the sentiment changes. The third task is generating a new sentence from the representation space. Our model uses a CNN for sentiment classification and sentence encoding. The encoded vectors are traversed using the MMD statistic and finally decoded using an RNN. An overview of the complete model is presented in figure 3.1. Each of the steps are given more detail in the sections below.

Figure 3.1: Overview of the algorithm and its different stages. An input sentence is represented as a matrix using word embeddings and given as input to a CNN. The CNN outputs a feature vector, $\mathbf{z}$, representing the sentence. This vector is moved in a semantic space using the MMD statistic. The traversal results in a new vector, $\mathbf{z}^*$, that should represent a sentence with the opposite sentiment. This vector is given as input to a text generating RNN which outputs the new sentence.

3.1 Sentiment classification and encoding sentences

A sentence is represented as a matrix where the rows correspond to the, 300-dimensional, *word2vec* [3] word embeddings for each word in the sentence. This matrix is given as input to a CNN, which is trained for binary sentiment classification. The CNN used in this project follows the work by Kim [2], with some modifications regarding filter sizes and number of filters. The network consists of one convolutional layer, one max-pooling layer and finally one fully connected feed forward layer. The filter sizes used for the convolutional layer were $1, 2, 3$ and 4 with 75 filters per size, resulting in 300 filters in total. The pooling layer threrfore outputs a 300-dimensional feature vector denoted $\mathbf{z}$. This feature vector is extracted from the CNN along with the predicted label, i.e positive or negative. The 300-dimensional feature vector is used as the encoding of the input sentence.

3.2 Manifold traversal of the representation space

Since the CNN is trained on binary sentiment classification, two separable distributions, one for positive and one for negative sentence representations, are generated. The MMD statistic, described in section 2.3, can be used to traverse a vector originating from one of these distributions to the other. The result of the traversal is a vector that resembles the encoding of a sentence with the opposite sentiment.

When moving the feature vector $\mathbf{z}$ by minimizing equation (2.12), the semantics of the original sentence may be lost if $\mathbf{z}$ is moved too far along the manifold. To control how far $\mathbf{z}$ is moved from its original location a *budget of change* [18], λ, is used. A source and a target set of sentence representations are created. The source set contains feature vectors for sentences with the same sentiment as $\mathbf{z}$ and the target set contains feature vectors for sentences with the opposite sentiment. A matrix $\mathbf{V} = [\mathbf{z}_1^t, \cdots, \mathbf{z}_n^t, \mathbf{z}_1^s, \cdots, \mathbf{z}_m^s, \mathbf{z}]$ is created from the target set $\mathbf{z}^t$, the source set $\mathbf{z}^s$ and $\mathbf{z}$. The traversed feature vector, $\mathbf{z}^*$, can then be expressed as $\mathbf{z}^* = \mathbf{z} + \mathbf{V}\delta$, where δ is the displacement of $\mathbf{z}$. Equation (2.12) can now be written as

$$f^*(\mathbf{z} + \mathbf{V}\delta) = \frac{1}{m}\sum_{i=1}^{m} k(\mathbf{z}_i^s, \mathbf{z} + \mathbf{V}\delta) - \frac{1}{n}\sum_{i=1}^{n} k(\mathbf{z}_i^t, \mathbf{z} + \mathbf{V}\delta), \tag{3.1}$$

where

$$\delta = \arg\min_{\delta} f^*(\mathbf{z} + \mathbf{V}\delta) + \lambda\|\mathbf{V}\delta\|^2, \qquad \lambda \in \mathbb{R}. \tag{3.2}$$

The minimization over δ uses the BFGS algorithm [22] described in section 2.4. The optimization of (3.2) is constrained by the budget of change, enforced in the last term.

3.3 Decoding sentences using a recurrent neural network

The traversed feature vector $\mathbf{z}^*$ is given as input to an RNN trained for generating text. In addition to $\mathbf{z}^*$, the RNN receives a start-of-sentence token as input in the first time step. For each time step, the RNN outputs the most probable word and feeds this word as input to the next time step. When the most probable word is an end-of-sentence token, the generation of words is terminated. The RNN consists of a single layer GRU cell, described in section 2.2.3.1, with a state size of 300. The weight matrix for the input, $\mathbf{W}_x$ in figure 2.5, consists of the 300-dimensional *word2vec* word embeddings for the words in the vocabulary.

3.4 Training the CNN and the RNN

When training the CNN on binary sentiment classification, the loss is calculated as the cross-entropy error between the predicted label and the true label for each sentence, see equation (2.9). Additionally, the CNN needs to encode information

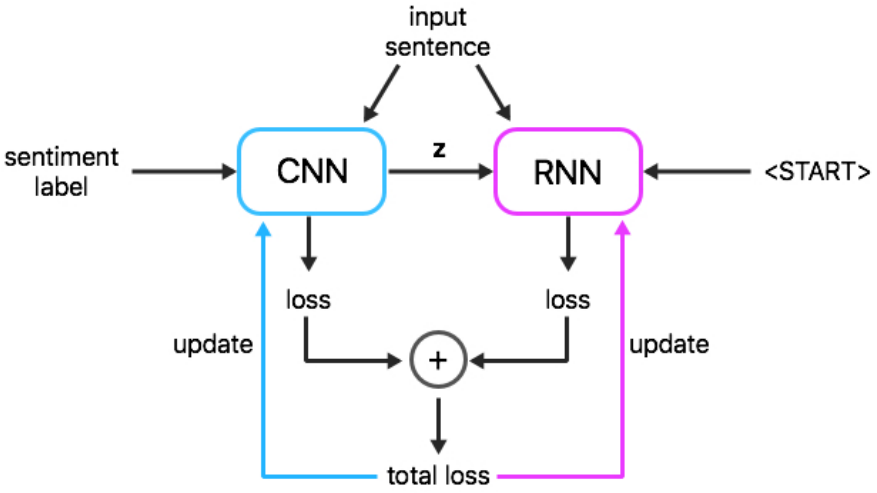
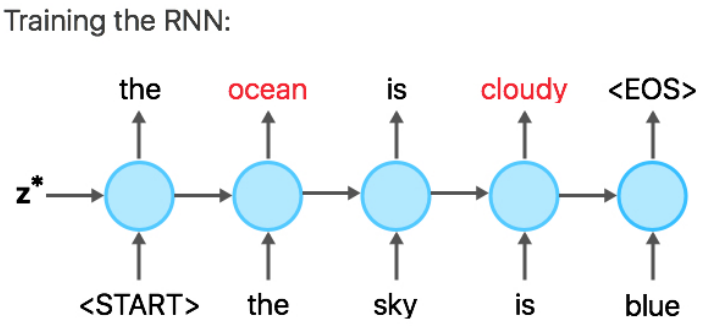


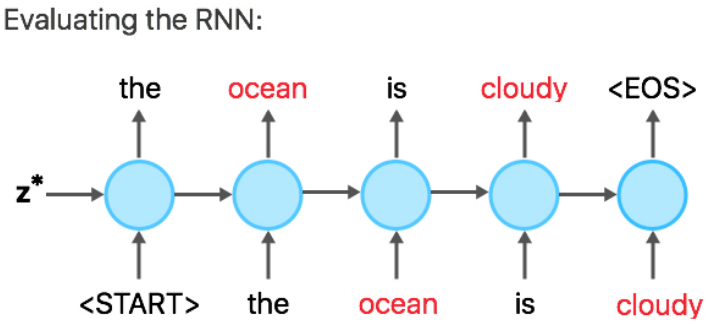
Figure 3.2: Training scheme for the CNN and RNN. The CNN takes a sentiment labeled sentence as input and produces a sentiment label and a feature vector **z** as output. The sentiment label is used when calculating the loss of the CNN and the feature vector is given as input to the RNN. In addition to the feature vector, the RNN takes the sentence (without a label) and a start-of-sentence token as input. The RNN produces a sentence as output and this sentence is used to calculate the loss of the RNN. Both networks are then updated using the unweighted sum of their respective losses.

about the topic and semantics of the sentence. Therefore the CNN is trained together with the RNN. During training, the feature vector produced by the CNN is given as input to the RNN and the loss for text generation is computed. These errors are added, producing a total loss which is used to update the weights in both networks. A schematic of the training procedure is illustrated in figure 3.2. In addition to the feature vector, the RNN takes a start-of-sentence token as input and is trained to generate the original sentence. The loss of the RNN is measured by calculating the cross-entropy error between the predicted word, $\hat{w}$, at time step t, in the generated sentence and the actual word, w, at the same time step from the original sentence, see equation (2.10). During training, at time step t the correct word from the previous time step $t-1$ is fed as input to the network. At evaluation time, however, the predicted word from the previous time step is used as input to the network in order to reduce the time it takes to train the neural network. Figure 3.3 highlights the differences in the input to the RNN during training and evaluation.

When the training of the CNN did not further improve performance, the CNN weights were locked and the RNN was retrained. The sentences from the training data were run through the trained CNN to create a set of feature vectors in 300 dimensions which served as input to the RNN. The training set was partitioned into batches of 64 sentences. All sentences shorter than 30 words were padded with "padding tokens" to the full length. The RNN was then trained to reproduce the original sentences over 14 epochs, given the sentence representations.



(a) During training, the RNN is given the correct word as input in each time step. Here, the RNN erroneously predicts the word "ocean" instead of "sky" and the word "cloudy" instead of "blue", but the correct words is used as input in the next time step.



(b) During evaluation, the RNN is given the predicted word from the previous time step as input in each time step. Here, the RNN erroneously predicts the word "ocean" instead of "sky" and the word "cloudy" instead of "blue", and uses the predicted words as input in the next time step.

Figure 3.3: Differences in the input to the RNN during training and evaluation. In this example, the correct sentence is "the sky is blue". During evaluation, previous erroneous predictions impact future predictions.

The CNN and the RNN were trained on three labelled data sets containing positive and negative sentences. The first data set is the movie review sentence polarity data set v1.0[1] (MR), introduced by Pang and Lee [7] in 2005. MR is a well known data set for sentiment analysis and consists of 10 662 labelled movie-review sentences from the movie review site www.rottentomatoes.com. The second data set is smaller and was introduced by Kotzias et. al [23] in 2015. It contains 500 reviews for cell phones and accessories from Amazon, 500 reviews for restaurants from Yelp and 500 movie reviews from IMDB[2]. Both data sets have equal amounts of positive and negative

[1]https://www.cs.cornell.edu/people/pabo/movie-review-data/
[2]https://archive.ics.uci.edu/ml/machine-learning-databases/00331/

sentences. The third data set is a subset of a data set[3] containing product reviews from various online sources, created by Täckström et al. [24]. In this data set, only the sentences that were either positive or negative were extracted, in total 923 positive and 1 320 negative sentences.

[3]https://github.com/oscartackstrom/sentence-sentiment-data

4
Experiments

The model, presented in the previous chapter, was implemented using the programming language Python 3. The TensorFlow [25] and NumPy [26] libraries were used for building the neural networks and the SciPy [27] library was used for optimizing the test statistic during the manifold traversal. The implemented model was evaluated with the goals for the project in mind, i.e reversing the sentiment of sentences while generating semantically similar and grammatically correct sentences. Each component of the model was tested separately and the complete model was tested as a whole. This chapter presents the experiments performed for evaluating the model. The results of the experiments are presented in chapter 5.

4.1 Accuracy of the neural networks

The manifold traversal is guided by the MMD statistic, relying on the existence of two distinguishable distributions of positive and negative sentence representations. Therefore, it is important that the classification accuracy of the CNN is high. Also, when generating sentences from the traversed vector, it is important that the RNN can decode the vector accurately. Thus, the RNN was evaluated on its ability to reproduce a sentence given a feature representation. The accuracy of the CNN and the RNN was therefore measured during training. The sentiment data set was randomly separated into a training set, containing 90% of the data, and a test set, containing the remaining 10%. The weights in the neural networks are only updated using the loss from the training set, not the test set. The training set was divided into batches of 64 sentences and the accuracy of both neural networks was evaluated on the test set periodically every 10th batch during training. The classification accuracy of the CNN was measured by simply calculating the percentage of correctly predicted positive or negative labels on the test set. In order to measure the accuracy of the RNN, the average per-word perplexity was calculated on both the training and test set. The perplexity can be interpreted as the number of words the RNN chooses between in each step and is calculated as e^{loss}. For the RNN, the cross-entropy loss was used, see equation (2.10).

4.2 Encoding sentiment and semantic content

In order to evaluate whether the encodings from the CNN contained information about sentiment and semantics, the feature vectors for the sentences were visualized for different sentiments and topics. The subset of feature vectors, used in each

experiment, was reduced from 300 to 2 dimensions using PCA, described in section 2.5. The visualizations were made using the first two principal components.

First, 1000 randomly sampled feature vectors, from the entire sentiment data set, which were correctly classified as either positive or negative by the CNN were visualized. Then, feature vectors with an additional topic label were visualized. This labelled data set was created through extracting sentences, containing specific words, from the original data set. Sentences containing "movie", "phone", "food", "comedy" or "drama" were extracted. The latter visualization was created in order to see how distinct topics ("movie", "phone" and "food"), as well as more similar topics ("drama" and "comedy"), were clustered.

4.3 Preserving semantic content during traversal

To assess whether the content in a sentence is preserved in the traversal, it was desirable to traverse and visualize sentences with distinct topics. The choice of topics was sentences containing either the word "phone" or "move", because such sentences would likely have little correlation in contrast to, for example, sentences containing either "comedy" or "drama". Negative sentences containing the word "movie" and positive sentences containing the word "phone" were traversed using different settings for the hyperparameters σ and λ. The optimization of the MMD was set up with 90 positive examples and 90 negative examples for the source and target sets. The examples consisted of an equal amount of sentences containing the word "movie" and sentences containing the word "phone". The topics of the sentences were not used for the traversal but needed when visualizing the results. For the visualization, the feature vectors for the examples and the traversed feature vectors were reduced to 2 dimensions, using PCA, similarly to the experiment described in section 4.2.

4.4 Evaluation of the complete model

There exists no single correct output for the manifold traversal, e.g given the negative sentence "The food did not taste well", both sentences "The food was amazing" and "I liked the food" are valid outputs that reverse the sentiment. Therefore, we used qualitative evaluation. The encoding-decoding, as well as the whole model, was evaluated by generating sentences from the feature vectors $\mathbf{z}$ (representing the original sentence) and $\mathbf{z}^*$ (the traversed vector) respectively. The generated sentences were manually compared to the original. We wanted the sentence generated from $\mathbf{z}$ to closely resemble the original sentence and the sentence generated from $\mathbf{z}^*$ to have the same context, but opposite sentiment, as the original sentence.

5
Results and discussion

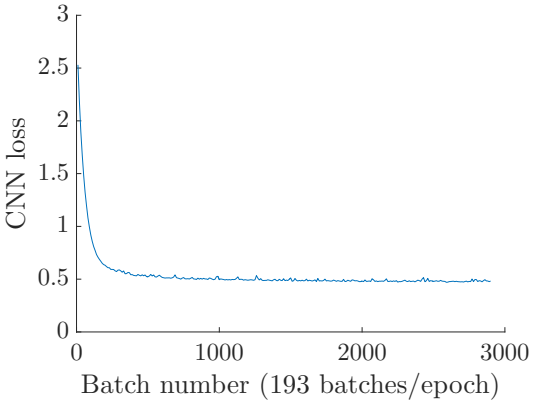
This chapter presents and discusses the results of the experiments from chapter 4. The implications of the results, the evaluation method of the complete model as well as design choices and data sets are also discussed. Some suggestions regarding how to improve the model are given, along with some thoughts about what can be done in the future.

5.1 Accuracy of the CNN and RNN

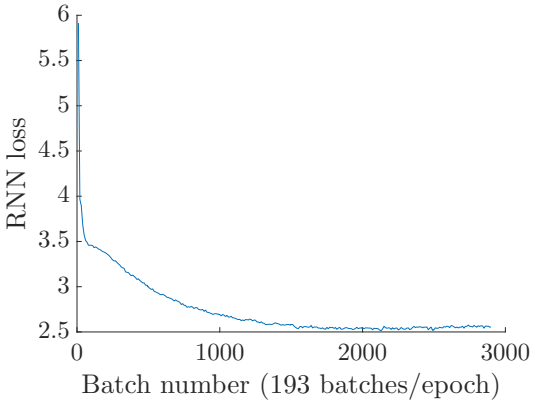
Training the CNN for sentiment classification, regarding the loss of the text-generating RNN, resulted in a classification accuracy of 80% on the test set. The evaluation procedure is described in section 4.1. The losses for the CNN and RNN were evaluated on the test set periodically during training and can be seen in figures 5.1a and 5.1b. The combined loss of the CNN and RNN can be seen in figure 5.1c. The classification accuracy was also evaluated periodically on the test set and is shown in figure 5.1d.

As figure 5.1a shows, the loss of the CNN decreases exponentially and converges towards 0.5. We found that using *l2*-regularization prevented the network from overfitting. Without regularisation the loss of the CNN started to increase after about 2000 batches of training. The accuracy of the CNN, seen in figure 5.1d, is very noisy after around 300-500 batches. This behaviour possibly occurs because the CNN has to take into account the loss of the RNN during training. In figure 5.1b we can see that the loss of the RNN is high and decreases relatively slowly between batch 100-300. This presumably introduces noise in the CNN accuracy because the CNN tries to adapt so that the RNN loss decreases. The noise is amplified because the learning rate is higher in the beginning of training.

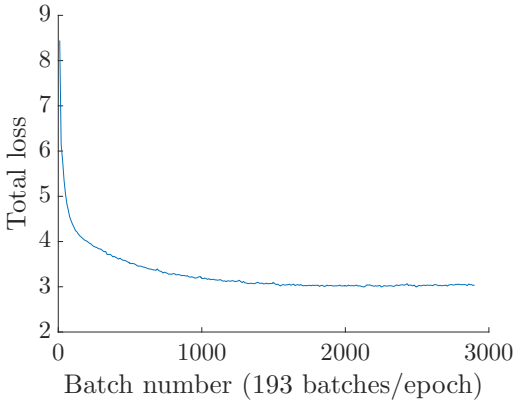
Figure 5.2 shows the perplexity for the RNN during its separate training (see section 3.4), computed for both the training and the test set. It is important to note that, during training, the RNN takes the correct word from time step t as input for time step $t + 1$, but during test the word with the highest probability at time step t is used as input for time step $t + 1$. The special procedure during training decreases the training time for the RNN.



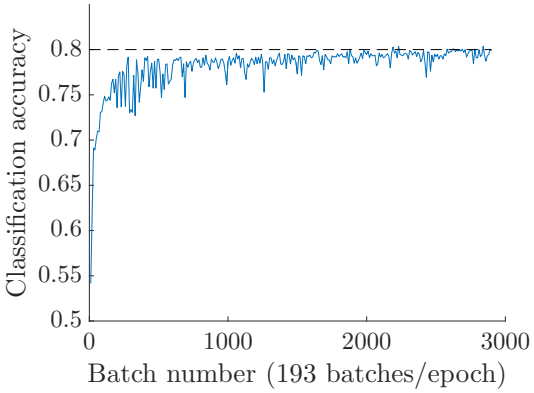
(a) Cross-entropy loss for the CNN, evaluated on the test set.



(b) Cross-entropy loss for the RNN during CNN training, evaluated when feeding states generated by the CNN.



(c) Total loss during training of the CNN.



(d) Classification accuracy for the CNN on the test set.

Figure 5.1: Loss and accuracy evaluated periodically on the test set during training of the CNN for sentiment classification. Subfigures 5.1a and 5.1d show loss and accuracy of the CNN for sentiment classification over 10 training epochs. Subfigure 5.1b shows the loss of the RNN for text generation when fed with states from the CNN during the training of the CNN and subfigure 5.1c shows the combined loss of the CNN and the RNN.

5.2 Encoding sentiment and semantic content

Figure 5.3, visualizes clustering of the dimensionality reduced feature vectors. The points are coloured based on their sentiment and, in figures 5.3b-5.3d, based on words they contain. It is, as expected, evident from all graphs that the CNN distinguishes positive from negative sentence representations as only the correctly classified sentences were used for the visualization. Additionally, the figures suggest that the CNN also makes a distinction between different topics.

Topics that are too similar like “comedy” and “drama” (figure 5.3d) seem to be hard

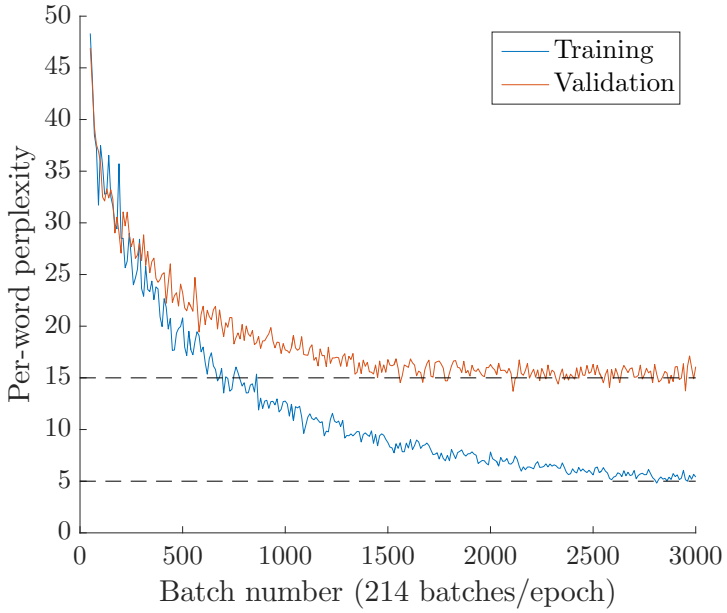
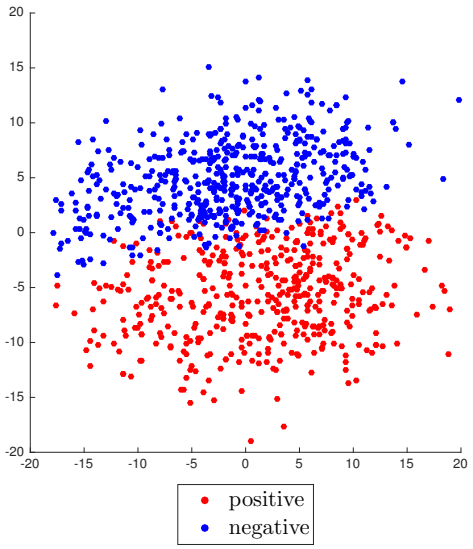


Figure 5.2: Perplexity of the RNN during training. The RNN was evaluated on the test set every 10 batches and the training perplexity was averaged over 10 batches.

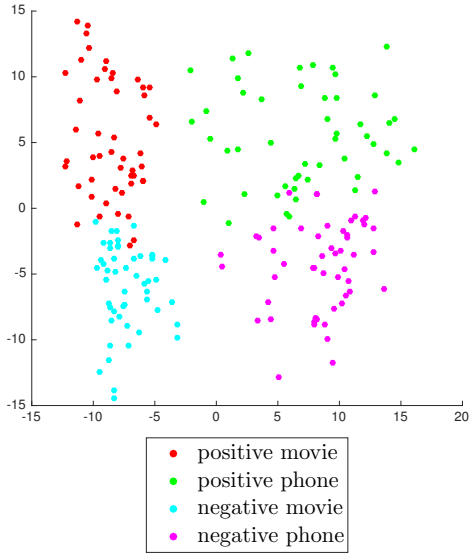
to distinguish in contrast to the topics "movie" and "food" (figure 5.3c) where we can see distinct clustering. Most likely, the sentences containing "drama" and the sentences containing "comedy" are related, and since we used the word2vec word embeddings the distance between the words is likely small. This might contribute to the similar encoding of sentences containing "comedy" and "drama". Another aspect to keep in mind is that the clusters are visualized using PCA. Since we plot the feature vectors in the space spanned by the first two principal components there may still exist a dimension, in which the variance is lower, but where sentences with "comedy" and "drama" are separated.

5.3 Preserving semantic content during traversal

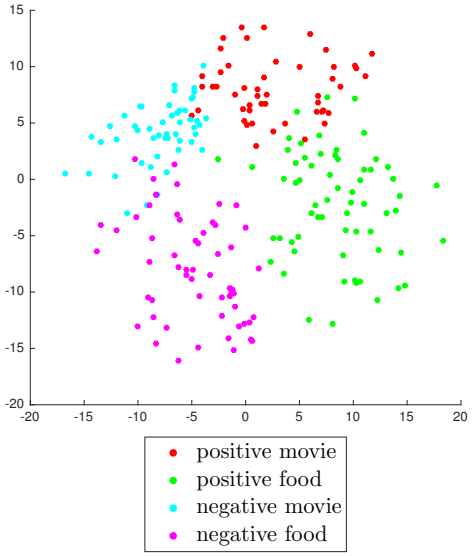
To assess whether the content in a sentence is preserved in the traversal, figure 5.4 shows how negative sentences containing the word "movie" and positive sentences containing the word "phone" are traversed using different settings of the hyperparameters σ and λ. The values for λ that were used and the resulting magnitude of the displacement vectors δ are listed in table 5.1. When traversing the manifold for the sentence containing the word movie, the displacement vector, δ, was initialized to a vector of zeros so that the traversal starts at the original feature vector. When traversing the manifold for the sentence containing the word phone, the displacement vector was initialized with uniform random numbers (between 0 and 0.02 for target indices and between -0.02 and 0 for source indices). The reason for the latter initialization of δ is that the optimal displacement should, in general, move away from the source vectors and towards the target vectors. The results in figure 5.4a show that a vector representing a negative sentence containing "movie" is moved



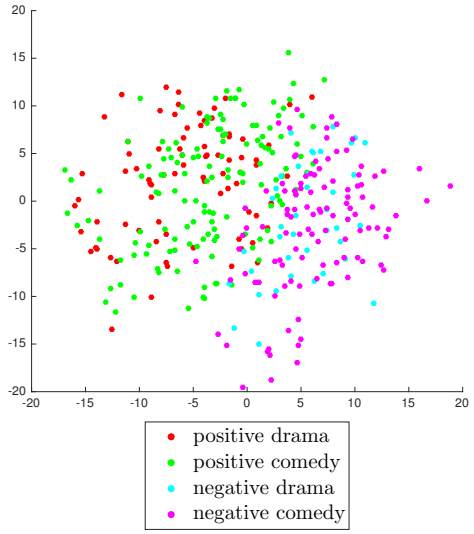
(a) Feature vectors for positive and negative sentences.



(b) Feature vectors for sentences containing either the word "movie" or "phone".



(c) Feature vectors for sentences containing either the word "movie" or "food".



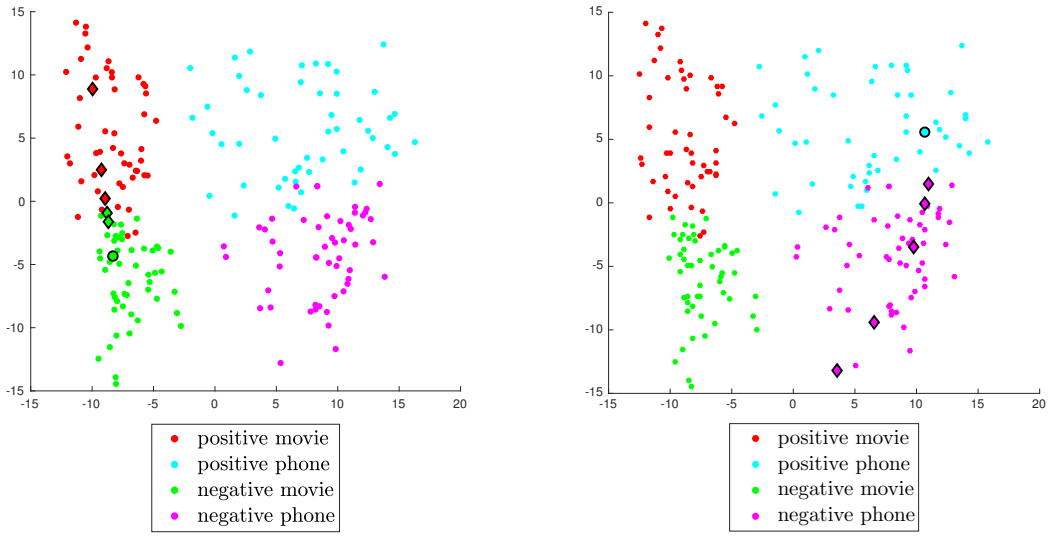
(d) Feature vectors for sentences containing either the word "comedy" or "drama".

Figure 5.3: Dimensionality reduction, using PCA, of feature vectors generated by the CNN.

so that the resulting vector lies within the cluster of positive sentences containing "movie". In the same way, we see in figure 5.4b how a vector representing a positive sentence containing "phone" is moved so that the resulting vector lies within the cluster of negative sentences containing "phone". This behaviour suggests that the context and semantics may be preserved during the manifold traversal. We can also see, in figure 5.4a, that the sentiment classification given by the CNN is not changed

when λ is too small. While reasonable, this result tells us that the way in which we enforce a budget of change actually may prevent the sentiment from changing. Since it is always desirable to change the sentiment, there could be a reason for relaxing this constraint in the optimization. Doing so might require the introduction of additional constraints, in order to preserve the semantics.

Since the manifold traversal is made using two sets of examples, source and target feature vectors, the traversed feature vector will to a large extent resemble the sentences in the target set. This means that if we traverse the manifold for a sentence with a different topic than the sentences in the source and target sets, the traversed vector might not preserve the topic of the original sentence.



(a) Traversal of the feature vector for a negative sentence containing the word "movie".

(b) Traversal of the feature vector for a positive sentence containing the word "phone".

Figure 5.4: Original and traversed feature vectors. Circles indicate the original vectors and diamonds indicate traversed vectors. The traversals were made using the different parameter settings presented in table 5.1.

Table 5.1: Different settings of the budget of change, λ, and the length of the resulting displacement, δ, for the manifold traversal presented in figure 5.4.

movie		phone	
λ	$\lvert\delta\rvert$	λ	$\lvert\delta\rvert$
$5 \cdot 10^{-4}$	2.842	$1 \cdot 10^{-3}$	4.088
$4 \cdot 10^{-4}$	3.552	$7.5 \cdot 10^{-4}$	5.747
$3 \cdot 10^{-4}$	4.731	$5 \cdot 10^{-4}$	9.684
$2 \cdot 10^{-4}$	7.062	$2.5 \cdot 10^{-4}$	17.970
$1 \cdot 10^{-4}$	13.614	$1 \cdot 10^{-4}$	24.227

5.4 Evaluation of the complete model

In table 5.2 some of the better examples of sentences generated by the trained RNN are shown. The original sentences originate from the data set and are generated before and after manifold traversal, as described in the experiment in section 4.4. The overall impression is that, while trained on the, very small, sentiment data set, the model works well in terms of changing sentiment. But, as figure 5.4 shows, the model fails to change the sentiment label if the budget of change is too small. We see that in some examples, the sentence generated from $\mathbf{z}$ is not very similar to the original sentence. In these examples, the sentence generated from $\mathbf{z}^*$ is more similar to the sentence generated from $\mathbf{z}$ than to the original sentence. This indicates that information is lost in the encoding-decoding procedure.

A more accurate encoder and decoder would require a larger set of data for training. The encoder would benefit from training on a more varied data set because, currently, sentences that have little relation to movie reviews are hard to encode. The decoder would benefit from training on a larger data set in order to improve on grammar. As the decoder does not require labelled sentences to train, the training data set can easily be extended. However, creating a larger sentiment labelled data set for the encoder is a project in itself. For the encoder and decoder to work well together it is desirable to use closely related data sets for their training.

After the traversal, the traversed vector is passed as input to the last layer in the CNN in order to make a sentiment prediction. Even though the sentiment classification from the CNN has changed to the desired label, it is, for some sentences, difficult to manually decide if the generated sentence has the opposite sentiment. This could be an effect from the problems in the decoding, discussed above. In table 5.2 we see that the generated sentences have the same topic as the original and that the generated sentences are composed by words similar to the original words. The decoder often generates expressions, of two or three words, like “just plain” and “very dissapointed”. The RNN has likely learned these expressions. We also found that shorter sentences were more easily encoded and decoded.

Table 5.2: Examples of sentences generated by the RNN, both from the original feature vector ($\mathbf{z}$) and from the traversed feature vector ($\mathbf{z}^*$), along with the original sentences.

Original:	the place was fairly clean but the food simply was n't worth it
From $\mathbf{z}$:	the food was pretty clean but it was n't worth the food place
From $\mathbf{z}^*$:	the food is also good food and also
Original:	the food, amazing
From $\mathbf{z}$:	food, food, amazing
From $\mathbf{z}^*$:	the food, which, just plain food
Original:	If you like a loud buzzing to override all your conversations, then this phone is for you!
From $\mathbf{z}$:	if you like a loud buzzing to override your conversations, then you is all for your phone to <UNK>
From $\mathbf{z}^*$:	the phone is great, especially the phone that is still a nice phone
Original:	...a roller-coaster ride of a movie
From $\mathbf{z}$:	a roller coaster of a movie coaster
From $\mathbf{z}^*$:	the movie is just a retread of garbage
Original:	it 's too bad the food is so damn generic
From $\mathbf{z}$:	it 's the food is so too bad the food is
From $\mathbf{z}^*$:	it 's the food is so nice food
Original:	a sharp and quick documentary that is funny and pithy, while illuminating an era of theatrical comedy that, while past, really isn't.
From $\mathbf{z}$:	a sharp, <UNK> comedy that is n't an engaging of theatrical, but this film and an interesting theatrical
From $\mathbf{z}^*$:	the whole of this is so awful that this just plain, even if this one this is just
Original:	...a delightfully unpredictable , hilarious comedy with wonderful performances that tug at your heart in ways that utterly transcend gender labels.
From $\mathbf{z}$:	a delightfully unpredictable, unpredictable that tug comedy with your heart in your face that at its heart performances comedy
From $\mathbf{z}^*$:	the title of this is so bad that they not even even if , even if this one
Original:	the food was excellent and service was very good
From $\mathbf{z}$:	service was great and the potatoes was great <UNK> <UNK>
From $\mathbf{z}^*$:	the whole was so bad and even the food was <UNK>
Original:	an ugly , revolting movie.
From $\mathbf{z}$:	an ugly , <UNK> movie
From $\mathbf{z}^*$:	an excellent , good movie experience
Original:	The movie was very interesting from beginning to the end.
From $\mathbf{z}$:	the movie was very interesting to the very interesting
From $\mathbf{z}^*$:	the movie was very disappointed the whole was very disappointed

5.5 Design choices and future work

The following sections discuss the motivation for the model design, data sets and evaluation metrics used in this project.

5.5.1 Model

The general idea of this project was to adopt the method presented by Gardner et al. [4] and examine whether a method used for traversing the manifold for images could be applied to text. As in [4], we chose to use a CNN as an encoder even though RNN encoders are more common in NLP. The reasons for choosing a CNN were the possibility of varying input lengths and because CNNs previously have been used for sentiment analysis [2], which was a part of the project.

The CNN encoder was tested separately before introducing the loss from the RNN into the error function for the CNN. We found that training the CNN separately resulted in a higher classification accuracy (84%). However, when training the RNN to decode sentences from the encoding from the CNN (trained without the RNN loss) we found that the encoding did not contain much information about the words that composed the input sentence. Hence, it was reasonable to train the CNN to regard the loss from the RNN.

Gardner et al. [4] did not use a neural network as decoder, instead they differentiated the encoding function and optimized over the input image. The reason why we chose an RNN as decoder was that RNNs are commonly used for generating text within NLP and have shown to perform well. It would have been possible to use a different decoder, for example a CNN. As pointed out in section 5.4, the RNN needs to be improved, for example by introducing a larger data set for training. Different configurations of the filter sizes and number of filters used for the CNN might also improve the decoder since theses parameters directly correlate to the RNN size.

In [4] the MMD statistic was used together with a Gaussian kernel function. Following their approach, the same statistic and kernel were used. It is possible to use another kernel such as a linear or polynomial kernel. In this work we use BFGS for the optimization of the MMD statistic, as in [4]. Since other methods were not considered, a different optimization method may improve speed or correctness of the traversal. When evaluating the objective function and optimizing δ, 90 positive and 90 negative examples were used. Using more examples would increase computation time but probably allow for making more subtle changes to the original vector $\mathbf{z}$.

5.5.2 Data set

The data sets used in this project consist of sentences written by people as reviews of products or movies. Because reviews posted on the Internet do not require proof reading, many sentences in the data set are subject to bad grammar and contain

slang words as well as misspelled words. When we created the vocabulary, only words occurring more than once were added and hence misspelled words and very uncommon words were replaced by the unknown-token. Since the model is trained to generate text that looks like text seen in the data set, it would produce better sentences if presented with a more well-written data set. Another problem with the data set is that it is not very diverse. A model trained on reviews can not be expected to produce, for example, narrative sentences or dialogues. Furthermore, the better part of the data set concern movies and cinematic experiences and it can therefore be difficult for the model to generate sentences with different subjects.

In future work, the model could be extended to take paragraphs, instead of sentences, as input. If the model can be trained on paragraphs, the *Large Movie Review Dataset* [28] can be used. This data set contains a total of 50000 positively or negatively labelled reviews. The RNN would likely benefit from using this larger data set as it adds context to sentences.

5.5.3 Evaluation metrics

To compare the performance of different machine learning algorithms on a specific task, it is desirable to have a qualitative or quantitative performance measure for the given task. Scores and measures used for other NLP tasks, like BLEU [29] for machine translation, are difficult to apply to the manifold traversal task since there exists no single correct output for each input. Consider for example, the sentence "I love music, it makes me wanna dance all night long", traversed into "Music sucks, I hate it". The new sentence has the opposite sentiment and tells something about music and would therefore be a valid output. This is also the reason why it is not possible to train the model end-to-end.

During evaluation, we used PCA to visualize the result as a complement to the manual evaluation of the generated sentences. Manual evaluation tends to be subjective if performed by only a few people and perhaps a survey would be a better alternative for evaluating the performance of the model. In a survey, a group of people are given a set of output sentences and are asked to grade the grammatical correctness and give a sentiment classification according to their own interpretation. In order to evaluate whether the sentiment has changed, an independent, preferably high-confidence, sentiment classifier could be used. However, we believe that the results seen in the visualizations, using PCA, support our conclusions about the model performance.

6
Conclusion

We introduce a model for reversing the sentiment of text through manifold traversal. The model encodes a sentence into a 300-dimensional feature vector, using a CNN that is trained for sentiment classification and sentence encoding. The feature vector is then traversed through a vector space guided by the MMD statistic, and a "budget of change" is used in order to constrain the displacement. The new vector is decoded into a sentence using an RNN decoder.

The CNN achieves an accuracy of 80% on the sentiment classification task. Visualizations, using PCA, show that the feature vectors contain information about both the topic, and sentiment, of the input. The results confirm that sentences can be represented in a semantic space and traversed in such a way that information about the sentiment changes but information about semantics and content is preserved. It is also possible to decode these representations and generate corresponding sentences. In order to improve the grammar of the output sentences, the RNN needs to be trained on a larger data set.

Future work might involve changing the sentiment of paragraphs, and not just sentences. Extending the model in this way would allow larger data sets to be used for training, which might improve the model. It would be interesting to evaluate different neural networks for encoding and decoding and compare the results.

Bibliography

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2013.

[2] Y. Kim, "Convolutional neural networks for sentence classification," *CoRR*, vol. abs/1408.5882, 2014.

[3] 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*, 2013, pp. 3111–3119.

[4] J. R. Gardner, M. J. Kusner, Y. Li, P. Upchurch, K. Q. Weinberger, and J. E. Hopcroft, "Deep manifold traversal: Changing labels with convolutional features," *CoRR*, vol. abs/1511.06421, 2015.

[5] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[6] A. Radford, R. Jozefowicz, and I. Sutskever, "Learning to generate reviews and discovering sentiment," 2017, cite arxiv:1704.01444.

[7] B. Pang and L. Lee, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales," in *Proceedings of ACL*, 2005, pp. 115–124.

[8] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[9] Z. Harris, "Distributional structure," *Word*, vol. 10, no. 23, pp. 146–162, 1954.

[10] M. Wahde, "Neural networks," in *Biologically Inspired Optimization Methods: An Introduction*. Ashurst Lodge, Ashurst, Southampton, SO40 7AA, UK: WIT Press, 2008, pp. 151 – 172, ISBN: 9781845641481.

[11] A. Severyn and A. Moschitti, "Twitter sentiment analysis with deep convolutional neural networks," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '15. New York, NY, USA: ACM, 2015, pp. 959–962.

[12] S. Poria, E. Cambria, and A. F. Gelbukh, "Deep convolutional neural network textual features and multiple kernel learning for utterance-level multimodal sentiment analysis." in *EMNLP*, 2015, pp. 2539–2544.

[13] J. Ebrahimi and D. Dou, "Chain based rnn for relation classification," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Denver, Colorado: Association for Computational Linguistics, May–June 2015, pp. 1244–1249.

[14] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013, pp. 6645–6649.

[15] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014.

[16] M. Sundermeyer, R. Schlüter, and H. Ney, "Lstm neural networks for language modeling." in *Interspeech*, 2012, pp. 194–197.

[17] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[18] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *Journal of Machine Learning Research*, vol. 13, no. Mar, pp. 723–773, 2012.

[19] C. T. Kelley, *Iterative methods for optimization*. SIAM, 1999.

[20] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[21] R. Bro and A. K. Smilde, "Principal component analysis," *Analytical Methods*, vol. 6, no. 9, pp. 2812–2831, 2014.

[22] R. Battiti, "Optimization methods for back-propagation: Automatic parameter tuning and faster convergence," in *International Joint Conference on Neural Networks*, vol. 1, 1990, pp. 593–596.

[23] D. Kotzias, M. Denil, N. de Freitas, and P. Smyth, "From group to individual labels using deep features," in *KDD*. ACM, 2015, pp. 597–606.

[24] O. Täckström and R. McDonald, "Discovering fine-grained sentiment with latent variable structured prediction models," in *Proceedings of the 33rd European Conference on Advances in Information Retrieval*, ser. ECIR'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 368–374.

[25] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.

[26] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: A structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[27] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–.

[28] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150.

[29] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ser. ACL '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 311–318. [Online]. Available: http://dx.doi.org/10.3115/1073083.1073135