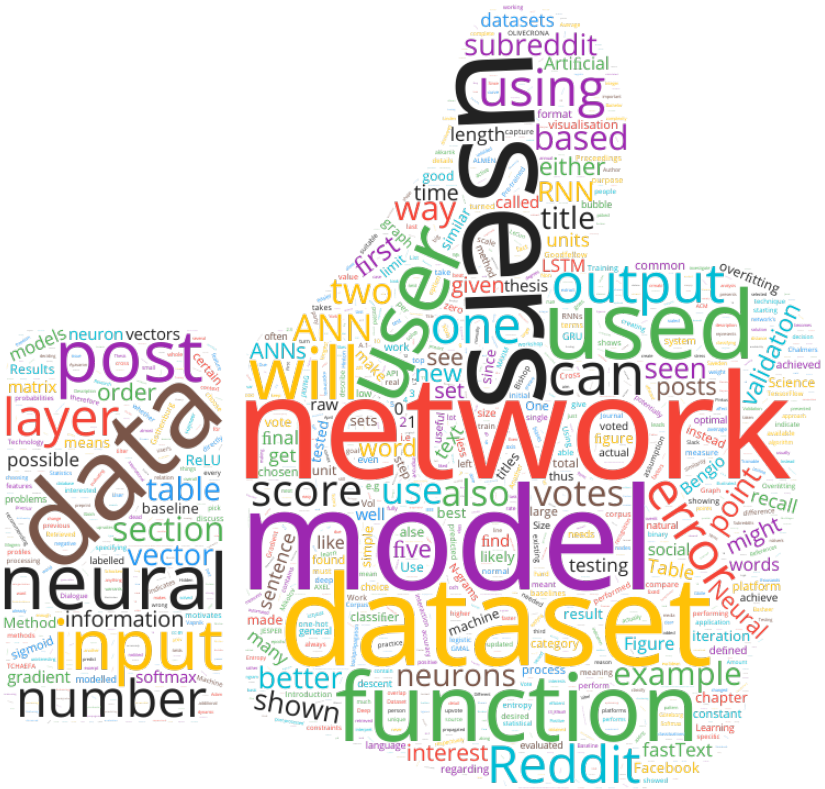
CHALMERS | UNIVERSITY OF GOTHENBURG



Recommending social platform content using deep learning

A reccurrent neural network model as an alternative to existing recommender systems

Bachelor of Science Thesis in Computer Science and Engineering

JESPER JAXING, ALEXANDER HÅKANSSON, MAXIM GORETSKYY,
GMAL TCHAEFA, AXEL OLIVECRONA, JONATAN ALMÉN

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

Bachelor of Science Thesis

Recommending social platform content using deep learning

A reccurrent neural network model as an alternative to existing recommender systems

JESPER JAXING
ALEXANDER HÅKANSSON
MAXIM GORETSKYY
GMAL TCHAEFA
AXEL OLIVECRONA
JONATAN ALMÉN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
University of Gothenburg

Göteborg, Sweden 2017

Recommending social platform content using deep learning
A reccurrent neural network model as an alternative to existing recommender systems

JESPER JAXING
ALEXANDER HÅKANSSON
MAXIM GORETSKYY
GMAL TCHAEFA
AXEL OLIVECRONA
JONATAN ALMÉN

Supervisor: Olof Mogren, Department of Computer Science and Engineering
Examiner: Richard Johansson, Department of Computer Science and Engineering

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gotehnburg
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31 772 1000

Cover illustration:
A word cloud representing the most common words in this thesis, in the shape of a hand. Generated using the free tool www.wordclouds.com

Department of Computer Science and Engineering
Göteborg, Sweden 2017

Recommending social platform content using deep learning
A reccurrent neural network model as an alternative to existing recommender systems

JESPER JAXING
ALEXANDER HÅKANSSON
MAXIM GORETSKYY
GMAL TCHAEFA
AXEL OLIVECRONA
JONATAN ALMÉN
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Bachelor of Science Thesis

Abstract

In this thesis a model based on artificial neural networks, seeing how they have found successes in a variety of fields in recent years, is proposed as an alternative to existing recommender systems. A recurrent neural network model is built for recommending content to users on the social forum Reddit based on the titles of posts. The model is compared against Facebook's fastText classifier and a model based on N-grams. The model proposed is performing close to, but not as well as either fastText or the N-grams based model. The model does not show any real advantages in its current state but a lot of potential improvements are proposed.

Keywords: Artificial Neural Networks, ANN, Recurrent Neural Networks, RNN, Recommender Systems, Automatic Recommendations, Text Classification

Sammanfattning

I denna rapport presenteras en modell baserad på artificiella neurala nätverk, då dessa haft stor framgång i flera olika områden de senaste åren, som ett alternativ till existerande rekommendationssystem. En modell som bygger på rekurrenta neurala nätverk byggdes för att rekommendera innehåll till användare på det sociala forumet Reddit baserat på inläggs titlar. Modellen jämförs med Facebooks fastText klassificerare och en modell baserad på N-grams. Den presenterade modellen presterar nästan lika väl som fastText och modellen baserad på N-grams. Modellen visar inte på några riktiga fördelar i sitt nuvarande läge men flera potentiella förbättringar presenteras.

Nyckelord: Artificiella Neurala Nätverk, ANN, Rekurrenta Neurala Nätverk, RNN, Rekommenderingssystem, Automatiska Rekommendationer, Text Klassificering

Acknowledgements

We would like to thank our supervisor Olof Mogren for the great insight and feedback that he has provided during the work with thesis, making it a fun process and enabling us to succeed. We would also like to thank the division for language and communication for their help with producing the writing of this thesis. Furthermore, the feedback we have received from other students have been of great help.

Contents

List of Figures

All figures used in this thesis are created by the authors themselves if nothing else is mentioned. These figures are shared under the CC-BY 4.0 License.

List of Tables

1
Introduction

In today's digital age it can be hard keeping track of all the content and conversations on different social platforms such as Slack, Facebook, WhatsApp and Reddit. People do not want to miss the conversations or news that might interest them, but rather get instant feedback when these conversations or news take place. However, the notifications of today are indiscriminate in that it is hard to get notified when something of interest is posted while not being notified too much. Some applications have tried to solve this by allowing manual *mentioning* or *highlighting* of a person to get their attention; Slack is one example of this [1]. However, these functions put a constraint on the users where the users themselves need to figure out who might be interested in some content. It is also deemed as bad manners to highlight a person – or a group of people – too often. Studies have shown that receiving an excessive amount of notifications can lead to stress [2].

One solution to this problem could be an algorithm that when given content from some social platform as input would output specific users that should be notified because they are likely to deem it interesting. An application using such an algorithm would relieve users of the stress caused by an abundance of notifications and contribute to an overall better experience for them. Even if the algorithm would not work perfectly, a lesser amount of irrelevant notifications could potentially make a big difference.

Similar problems to the one mentioned above have been solved in other fields such as electronic commerce. Amazon uses a technique called recommender systems to recommend new items for their users [3]. Their recommender system is not based on artificial neural networks but rather compares the similarity between new products and products that a user has previously bought or rated [3]. This thesis will evaluate if it is possible to lessen irrelevant notifications by using an artificial neural network. Artificial neural networks are generally better at pattern recognition and clustering compared to other statistical methods [4] which motivates their use in this thesis to classify what people are interested in. When working with natural language in the form of text, as in this project, it is common to use a specific sort of network called recurrent neural network (see details in 2.6). Therefore this thesis will specifically evaluate the possibility of using a recurrent neural network to lessen irrelevant notifications.

1.1 Purpose

The goal of this thesis is to evaluate the performance of using recurrent neural networks to personalise recommendations of text content for users on social platforms.

1.2 Problem Description

To complete the goal of this project, the relation between some content on a social platform and users' interest needs to be modelled. An artificial neural network model will be used and evaluated for this purpose. To successfully model the problem, there are some factors that must be considered.

First of all, in order to use an artificial neural network, a dataset which can be used for training is needed. When a dataset is obtained, the artificial neural network must be modelled to capture the relation between the contents of a text and a user's interest. The modelling is a key part of the problem where a lot of experimentation with different parameters is needed to achieve adequate results. Once the best result, given time constraints, has been achieved the model also needs to be compared with some baseline. This is to evaluate whether the model is good in practice or not. These tasks, as described in more detail below, must be addressed and completed in order to achieve the goal of the project.

1.2.1 Selecting a Dataset

There are many public datasets to choose from, however a dataset needs to satisfy the following constraints for the proposed model to be able to learn from it:

- It must be sufficiently large
- Each data point needs information that can be used to determine if the predicted result was correct or not – It must be labelled

As part of this project, potential datasets that fulfilled the above constraints were examined to select the most suitable one.

1.2.2 Finding Hyperparameters

Hyperparameters, such as the model's shape, size and how it computes its results, determine the characteristics of the artificial neural network. The chosen hyperparameters for the model will, in the end, affect its performance. Some of the hyperparameters that were examined are described below:

- Depth of network, how many layers the network needs
- Width of layers, how many nodes each layer has
- Size of embedding matrix
- Learning rate

More explanation of artificial neural networks and hyperparameters can be found in Chapter 2 & 3.

1.2.3 Comparing with a Baseline

When evaluating the performance of the network one or more baselines are needed as reference points. How well the resulting model compares against existing methods will give an indication of whether using artificial neural networks could be a good choice for solving the problem or not. This comparison fulfils the project's purpose of evaluating the performance of artificial neural networks for the described problem. The baselines will be compared with the model in terms of the statistical measure F_1 score (see Equation 3.3) when giving recommendations.

1.3 Scope

This thesis will develop an artificial neural network model, and evaluate the possibility of using it to accurately recommend text content to users of a social platform such that it will be interesting to them.

1.3.1 Limitations

This project will mostly be theoretical and will not focus on business applications to the problem.

Due to technical limitations regarding computing power and available data, the number of users that will be considered when making recommendations will be smaller than what would likely be desired in a commercial implementation. This is to make the training of the network take less time and thereby make it possible to complete more experiments. The total number of users that will be considered when making recommendations will be fixed to the number of users in the chosen dataset.

One issue that arises when dealing with data from social platforms is the ethical concern regarding integrity. It is not hard to imagine that a system that could tell whether or not something is of interest to a specific person could be abused. The system could perhaps discover areas of interest that a person would have wished to keep private, this is one example of potentially many problems. Although this is an important issue, the thesis will not cover this subject.

2

Artificial Neural Networks

This chapter will discuss the basics of artificial neural networks, which will be referenced throughout the rest of the thesis. All of the theory addressed in this chapter have been used in some way to achieve the results presented in Chapter 4. This chapter will not introduce anything new to someone who is familiar with artificial neural networks.

2.1 Introduction to Artificial Neural Networks

An artificial neural network is a machine learning model inspired by biological neural networks [5]. Goodfellow and his collegues [6] describe artificial neural networks as being directed and acyclic graphs (see Figure 2.1). They go on to say that the network is constructed of artificial neurons which are structured in layers – these neurons are represented as nodes in a graph. Each neuron represents a function that takes some input from preceding nodes and produces an output. The input data to the network is represented by a vector $\mathbf{x} = [\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \ldots, \mathbf{x_n}]^T$ with n features. The vector $\mathbf{x}$ could, for example, represent a whole sentence where vector $\mathbf{x_i}$ represents a word.

The edges in the network are connections between pairs of neurons. Each connection between two neurons has a weight. These weights will change as the network is optimised, as described in Section 2.5, and the way they change will directly impact the network's output. Updating the weights between neurons is thus a key component of how an artificial neural network works. In practice, this means that the weights should be updated so that the network gradually produces an output more similar to the desired target.

The input to a layer in the network is the matrix product between the output from a preceding layer and the weights between those two layers (usually with some bias term added as well). The weights are used to transform the input before it is propagated to subsequent layers. A convenient way to represent weights between two layers j and k is by a matrix, W_{jk}, where each element is the weight between a neuron in layer k and a neuron in layer j, where $k < j$. Let $\mathbf{z_j}$ denote the input to layer j where W_{jk} are weights to layer j from layer k and $\mathbf{x_k}$ is the output from layer k. Let vector $\mathbf{b}$ be a bias term that is added for more degrees of freedom. The input to layer j is thus as shown in Equation 2.1.

$$\mathbf{z_j} = \mathbf{W_{jk}} \cdot \mathbf{x_k} + \mathbf{b_j} \tag{2.1}$$

Let $\mathbf{y_i}$ denote the output of the layer i. For the first layer in the network (the input layer), $\mathbf{y_1} = \mathbf{x}$ where $\mathbf{x}$ is just the input data. The output, $\mathbf{y_i}$, of a layer i is the result of applying some activation function on the input to the layer, $\mathbf{z_i}$, as shown in Equation 2.2. Activation functions are described in further detail in Section 2.3 but can, in summary, be seen as a way to scale the output element-wise.

$$\mathbf{y}_i = f(\mathbf{z}_i) \tag{2.2}$$

During training, once the data have been propagated to the last layer, an error function will be used to measure the error of the model by comparing the final prediction with the expected output for the given input. The details about error functions can be found in Section 2.4. Furthermore, the results of the error functions are used when training the network, which is explained in Section 2.5.

Depth and width are often used to describe the shape of an artificial neural network [6]. The depth determines how many layers the network contains. The layers between input and output layer are called hidden layers. The more neurons a layer has, the wider it is considered. [6], state that the reason for using multiple layers are taken from research in neuroscience. A large network (in either dimension) has more degrees of freedom which generally means that it will fit the data better, but could overfit (see Section 2.5.3).

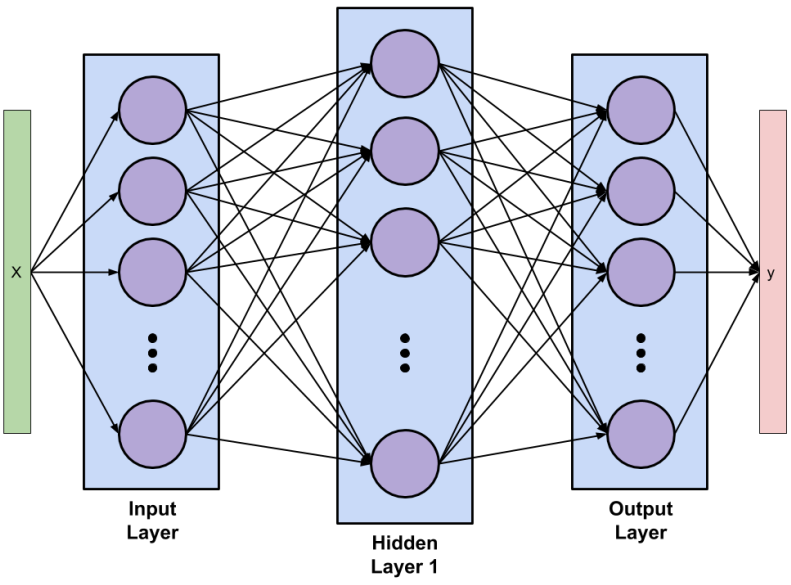


Figure 2.1: A graph visualisation of a simple artificial neural network. $\mathbf{x}$ and $\mathbf{y}$ are input and output vectors respectively

2.2 Uses of Artificial Neural Networks

A common use of machine learning is classification. It is the task of classifying a given data point as belonging to one (or more) of k sets, classes, based on previously observed data [7]. Classification can be visualised as a graph of data points with curves separating k classes, these curves are also referred to as decision boundaries. For the purpose of this thesis, an artificial neural network will be used for recommending text content to users. There are however a lot of different machine learning techniques which solves the problem of automatically classifying data, e.g. Naïve Bayes Classifier [8], Support Vector Machines (SVM) [9], [10], or Decision Tree Classifier [11].

Classification of data typically requires extraction of features that can be compared. Feature engineering, as this is called, is a process that usually requires extensive domain knowledge. When using artificial neural networks, the process of feature engineering can be omitted, thus making it easier to model more complex problems [12], [13]. This property of artificial neural networks makes them suitable for many tasks. One example is to model sequences of text as it is not necessary to manually find features in the text that could be used for classification, which motivates their usage in this project.

2.3 Activation Functions

The activation function scales the output from a node to create a new signal that is sent to the next layer [4]. By using non-linear functions, it is possible to scale the output and find non-linear decision boundaries which allow better fitting to the data [4], [5]. Different activation functions can be used for different purposes in the same artificial neural network, it is not necessary to choose only one. In this project, a few common activation functions have been used, and are described below.

2.3.1 Logistic Function

The logistic function is a non-linear function with an *S* shaped curve, as defined by Equation 2.3, that squashes the input, z_j from Equation 2.1, between $[0, 1]$.

$$f(\mathbf{x}) = \frac{1}{1 + exp(-\mathbf{x})} \tag{2.3}$$

In this project, the logistic function has been used in the final layer of the model as the output function as it is possible to interpret its output as probabilities. This particular function is especially useful for multi-label classifications (where some input can be classified as belonging to more than one class, e.g. being interesting to more than one users). When using the logistic function like this, it is common to use the negative log-likelihood error function (see Section 2.7) since this is particularly useful for multi-label classifications [14].

2.3.2 Rectified Linear Unit Function

The Rectified Linear Unit (ReLU) function is a non-linear function defined in Equation 2.4. This function has the benefit of being unbounded as opposed to the logistic function, which ranges in the interval $[0, 1]$. The ReLU function has become popular in the past few years due to it being less susceptible to vanishing gradient compared to other activation functions [13], [15].

$$f(x) = max(0, x) \tag{2.4}$$

The ReLU function was used as the activation function in the hidden layers of the model.

2.3.3 Softmax Function

The softmax function is defined in Equation 2.5 where $\mathbf{x}$ is a vector of n elements. Softmax scales the vector entries to be between $[0, 1]$, while normalising them all to sum to 1.

$$f(\mathbf{x}_j) = \frac{e^{\mathbf{x}_j}}{\sum_{i=1}^{n} e^{\mathbf{x}_i}} \tag{2.5}$$

When using the softmax function in the final layer of the network to create an output it has been shown that using the cross entropy error function together with softmax gives a better accuracy compared to other error functions [16], [17]. By using the softmax activation function, the output can be interpreted as normalised probabilities between the n output classes. Because of the normalisation, this becomes useful for classification where there is only one correct class. Softmax is used during pre-training of the network and tested as an output function for the whole model (see Section 3.3.2).

2.4 Error Functions

The error functions in artificial neural networks are used to compare the network's predicted output with the correct output, for a given input. When training the network, the difference between the predicted and correct output is used to minimise the error (more details in Section 2.5). There are primarily two different error functions that are used and evaluated in this thesis. For the error functions described below, y_k indicates the k^{th} prediction of the network, t_k indicates the correct value of the k^{th} prediction, and K is the number of classes.

One of the error functions examined is the cross entropy error function. It is commonly used when working with artificial neural networks because of its overall performance, compared to alternatives, when performing backpropagation (see Section 2.5.2 for details on backpropagation). The cross entropy function is defined in Equation 2.6 [18].

$$E_K = -\sum_{k=1}^{K} [t_k \cdot ln(y_k) + (1 - t_k) ln(1 - y_k)] \tag{2.6}$$

The reason behind the cross entropy function's popularity is that the derivative with respect to the weights is proportional to the difference between the predicted value and actual value, leading to better performance of the network [19].

The other error function that is evaluated is the Negative Log-Likelihood (sometimes called multi-class cross entropy) function [20]. This error function, defined in Equation 2.7 [21], could be interpreted as a more general version of the cross cntropy function. It is more suitable for multi-label classification problems compared to the Cross Entropy function which is more suitable for single-label classifications.

$$E_K = \mathbf{t} - log(\mathbf{y}) + (1 - \mathbf{t}) - log(1 - \mathbf{y}) \tag{2.7}$$

In this project the Negative Log-Likelihood function is used as the error function when using the logistic function (see Equation 2.3) as output function for the network.

2.5 Training and Optimisation

Training and optimisation in machine learning is the process of learning from data. The way a certain machine learning model learns from data usually differs, but the training of an artificial neural network can be seen as a problem of optimising the network's weights. For this to work, an error function is needed that determines how much a certain prediction for a data point is wrong compared to the true classification of that data point. The optimisation objective is to find the weights and biases in the artificial neural network that minimises the error for the training data. This can be achieved by applying backpropagation and some optimisation method, e.g. Gradient descent (see Section 2.5.1). The process of learning from some previously classified data is called supervised learning [13]. This training process is required to improve the performance of the artificial neural network [6].

2.5.1 Gradient Descent Optimisation

Gradient descent is an iterative algorithm where the gradient of a function is calculated in steps. The computed gradient is used to move towards the minimum of the function. This is repeated until a local or global minimum of the function is found. Computing the gradient for the whole dataset is computationally heavy. This leads to that the datasets used for training are divided into small batches, minibatches. When using minibatches, the gradients of the error function for each data point in the minibatch is computed and then averaged to approximate the full gradient. When applying gradient descent to this approximation it is known as stochastic gradient descent. It has been shown that stochastic gradient descent is fast for convex functions [22]. There are no guarantees that an artificial neural network will be convex, but there is a specialised and efficient implementation of stochastic gradient descent called *Adam* [23] that is commonly used for artificial neural networks. The Adam optimiser is used in this project to optimise the weights and biases of the artificial neural network model to give a minimal error. By minimising the error the predictions of the network should improve.

2.5.2 Using Backpropagation

Backpropagation is a method used for training artificial neural networks. When you feed an input vector (e.g. a sentence) to the network, the network will propagate the vector forward until it reaches the last layer and presents a prediction. The prediction is then compared to the desired result for the given input via an error function to determine how well the network performed. The error is propagated backwards through the network to determine how much every individual weight and bias impacted it. The weights and biases for each neuron are then updated accordingly. How the weights and biases are updated depends on which optimisation method is used. This training is performed on all of the data points in a training data. Each training iteration over the complete set of data is called an epoch.

2.5.3 Overfitting

Overfitting can be described by a state of the model where the network does not generalise well, meaning that the network is too specialised on the training data. This phenomenon results in bad accuracy of the network when it is faced with previously unseen data. This is a common problem if the network has too many parameters that can describe the training data instead of capturing the general idea of the data.

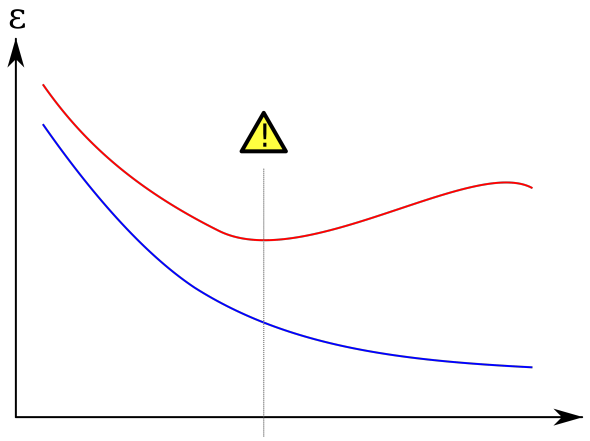


Figure 2.2: An example of overfitting. The red line shows the error on the validation data and the blue line shows the error on the training data. It is desired that the error is as low as possible. "Overfitting" by Gringer, edited by Dake, is licensed under CC-BY 3.0.

As seen in Figure 2.2 the network is performing better and better for both training and validation sets. However, it reaches a point where the validation error starts going up. That point is where the overfitting has occurred. The training error is still decreasing meaning that the network keeps learning on the training data instead of generalising from it.

2.5.4 Regularisation

Regularisation is a set of techniques used in order to prevent the overfitting problem described in Section 2.5.3. These techniques increase the performance of the network as they allow the network to continue generalising without overfitting, at the cost of sometimes limiting the networks. In this thesis two different regularisation techniques have been used:

- Dropout
- l^2-loss regularisation

Dropout is a rather newly discovered regularisation technique that prevents overfitting by randomly disabling neurons in the artificial neural network [24]. During the training of the network, neurons are randomly disabled with a probability p. This means that the network has to teach other neurons to do the generalisation that the disabled neuron previously did. It is important to note that this is only used during the training of the network. When validating the performance of the network or using the network all neurons are used.

The purpose of l^2-loss regularisation is to prevent weights from becoming extremely large by penalising them based on their size. The l^2 regularisation is used by adding the l^2-norm of every weight in the artificial neural network to the error function. The penalisation leads to a limitation in the network's capacity, or *statistical complexity*, in terms of the l^2-norm [25].

A powerful aspect of these two regularisation techniques is that they work alongside each other. This is because they are implemented in different parts of the artificial neural network model.

2.6 Recurrent Neural Networks

A recurrent neural network is an artificial neural network that takes context into account. It accounts for the context in the sense that the current output is dependent on the previous. This context dependency is often depicted as an artificial neural network that has its own output as input, but in practice, recurrent neural networks are instead modelled as chained (or *unfolded*) *units* as shown in Figure 2.3. A unit takes two inputs; the output from the previous unit and input data. The output produced is passed along to the next unit but could also be retrieved and used. An arbitrary number of units can be chained in this way, but in this project the number of units will be related to the length of the input text.

Recurrent neural networks are particularly useful when modelling with natural language as input since words in a sentence often are dependent on what comes before them. When using sentences as input, it is common to have one word as input per unit in the recurrent neural network [26]. This is how recurrent neural networks were used in this project to model social platform content.

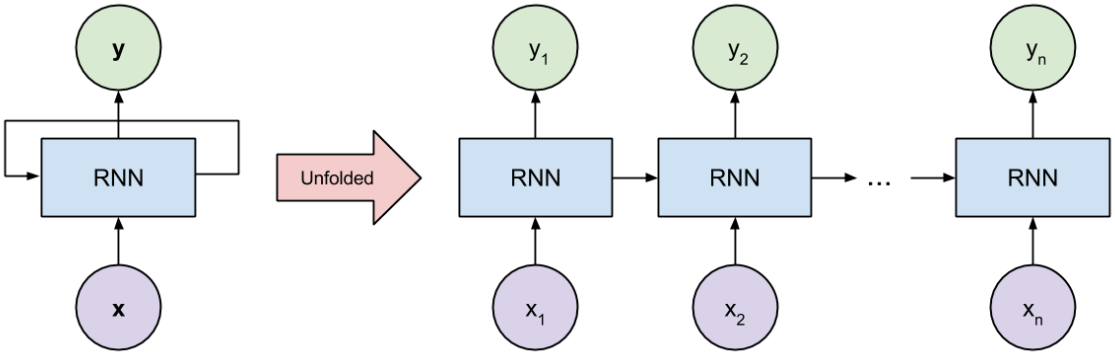


Figure 2.3: A visualisation of a recurrent neural network layer and how it is unfolded. $\mathbf{x}$ and $\mathbf{y}$ are the input and output vector respectively

The units in a recurrent neural network can be implemented in different ways. Popular unit implementations are Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU). The LSTM unit was introduced to solve the problems of inefficiency in the backpropagation for recurrent neural networks [27]. The inefficiencies occurred because the error gradient would decay and approach zero, making training very slow, and this is what the LSTM solves [28]. The GRU unit is a fairly recent addition to the field, and it is comparable to the LSTM in accuracy, but the dataset used can have a big impact on which one performs better [29]. The fact that either of the unit implementations might perform better based on the dataset used motivates testing them both in this thesis.

2.6.1 Using Recurrent Neural Networks for Natural Language Processing

As previously mentioned, when working with natural language it is common to let every unit in a recurrent neural network layer take one word as input. This is illustrated in Figure 2.4 below.

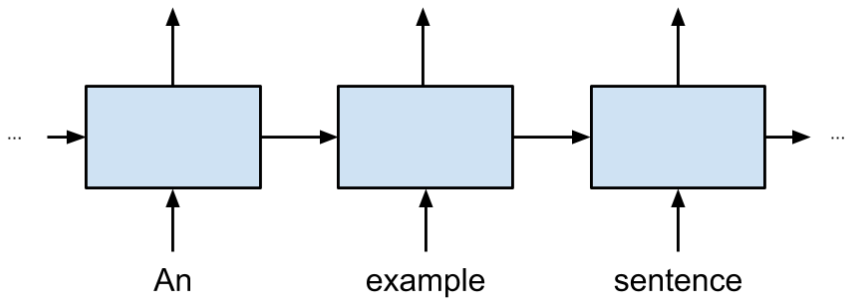


Figure 2.4: A simplification of how a recurrent neural network layers process natural language as input

The units, however, just as an artificial neural network, expect some vector as input. This means that the input sentences have to be transformed into vectors. A common way to achieve this is to use indices instead of strings of words. With this approach, each unique word is given a unique number, and an input sentence then becomes a vector of word indices. It is also common to take the transformation one step further and turn each word index into a so-called one-hot vector [30]. Using one-hot vectors, each word is represented as a vector $\mathbf{w}$ of length C, where C represents the total number of unique words, having a 1 at the position of the given word's index and zeros everywhere else (this is illustrated by Table 2.1). Since the length of the vector is fixed, it means that new words that are not in the known set of words, the vocabulary, can not be represented as a one-hot vector – to solve this, new words are represented together as *unknown* (usually with an index of 0).

Word	Vector representation
This	$[0, 0, 0, 0, 1]^T$
is	$[0, 0, 1, 0, 0]^T$
a	$[0, 1, 0, 0, 0]^T$
normal	$[0, 0, 0, 1, 0]^T$
sentence	$[1, 0, 0, 0, 0]^T$

Table 2.1: An illustration of the sentence "This is a normal sentence" using one-hot vectors, given that the vocabulary is {*unknown*, a, is, normal, this}.

Another problem with one-hot vectors is that they do not hold any information about relationships between words. This motivates using another representation for words in order to capture the connection. This can be done with algorithms such as word2vec [31] or GloVe [32], which produce so-called *embedding matrices*.

3
Method

This chapter describes what has been done and discusses the decisions taken.

3.1 Deciding on a Dataset

As described in the initial problem description, deciding on a dataset was essential for the project to proceed. A first step was thus to find a dataset which satisfied the constraints presented in Section 1.2.1. This was needed as a first step since an artificial neural network needs a source of data to learn from. Since artificial neural networks are mostly used for classification, the data used to train it must be labelled. It thus saves a lot of time if the chosen dataset is already labelled when fetching it, or that it can be automatically labelled by some automated process. Manually labelling several thousands of data points was never an option when deciding on the dataset. It would be too time-consuming for the scope of the project.

After searching for different sources of data, it finally came down to two datasets; Ubuntu Dialogue Corpus [33] and a dataset based on posts with corresponding user votes from Reddit (see appendix A). The Ubuntu Dialogue Corpus contained forum posts with back and forth discussions, whereas the Reddit dataset contained posts and whether certain users had up- or down-voted those posts. Using the Reddit API (`https://www.reddit.com/dev/api/`) it was also possible to extract more information from a post, such as which subreddit (category) it belonged to.

The reason for having the Ubuntu Dialogue Corpus and the Reddit up- and down-votes data as final considerations comes down to a few factors. Most important were the overall characteristics of the data, i.e. the data was text content written by users for other users. There was also interaction with the content which could be used to indicate interest in a topic. The availability of the data, its format, and its size were also factors.

After comparing the two datasets the Reddit dataset was chosen. It had a simple format and primarily it had a stronger indication of user interest compared to the Ubuntu Dialogue Corpus. In the Reddit dataset, content was directly linked to users on the platform which had either up- or down-voted it. On Reddit, up- or down-voting content means that the user clicks a button indicating if they like or dislike a certain post. An assumption was made that the active decision to either up- or down-vote some content would indicate interest and that an average user is less likely to engage

in a full discussion (as in the Ubuntu Dialogue Corpus) compared to just clicking a button. Given time constraints, the thesis only evaluates the case with the assumption that all votes, positive or negative, represent interest in the content and a lack of interaction with a post is interpreted as disinterest in it.

3.2 Gathering Data

The Reddit dataset is not so useful in its raw form. The raw data only consisted of ID references, as seen in Table 3.1. This data had to be converted for this project, and the result can be seen in Table 3.2.

Username	Post ID	Vote
2bornot2b	t3_89ko9	-1

Table 3.1: An example data point in the raw Reddit dataset showing that the user 2bornot2b has downvoted the post with ID t3_89ko9.

Title	Content	Subreddit	Upvotes	Downvotes
A Graph of NP-complete Problems	*<empty>*	compsci	izzycat, cypherx, HattoriHanzo	*<empty>*

Table 3.2: An example data point in the processed Reddit dataset showing information about a post (with no content) and which users showed interest in it.

In the raw form shown in Table 3.1 the data is modelled as a relation from a specific user to a specific post. A post is also represented as a single ID instead of its title, content and subreddit. To get from the raw format to the processed format two steps were made.

Firstly, the ID of a post was used to retrieve more meaningful content from the Reddit API. This included getting a post's title, subreddit and eventual content (a lot of posts only had a title with no other content). As a post ID is only a reference in Reddit's database, the API was used to extract more information by passing the ID to it. Specifically, the API had an endpoint that could take one or more post IDs as input and return a list with all information available about the posts with the specified IDs. This enables the extraction of the posts' titles and content as well as which subreddit they belong to. As the dataset is large it would have been ineffective to make the extraction from the post IDs manually, therefore the extraction task was automated. The source code for this automation program is available at `https://github.com/recnet/reddit-scraper`.

Secondly, the data were re-arranged to model a relationship from a unique post to several users. In the original format, each data point models a single user's vote for a post. A data point in the new format contained the information of a post and all users who have voted on that post. The final format, as shown in Table 3.2, was achieved by loading all of the data into a relational database. By having the data in a relational database, it was easy to extract different samples of data suitable for the experiments performed. As an example, it could be used to split the dataset into training, validation and testing subsets. It could also easily be analysed from a statistical standpoint in order to find patterns that could impact the result.

3.3 Modelling the Artificial Neural Network

The general characteristic of the problem is that given some post, one or more users should be recommended based on their previous interest in other posts. A Reddit post in the given dataset is described by three properties; a title, some content and a subreddit. This means that the artificial neural network model should be able to use natural language as input. This property of the problem motivates the use of a recurrent neural network (see Section 2.6) to fully capture the input of natural language. The models described in this project were developed and tested using the machine learning framework *TensorFlow* [34].

3.3.1 A Basic Starting Point

The initial model was implemented to have an artificial neural network with a recurrent neural network as the input layer, no hidden layers and an output layer. For the output layer, each user had to be represented in a way that made it possible to compare them against each other in terms of how likely they were to be interested in a given input post. In practice, this meant that the output layer should scale the output of the model between $[0, 1]$, as this allows for interpretation as probabilities. Two activation functions which do this were considered; the softmax function (see Section 2.3.3) and the logistic function (see Section 2.3.1). As the problem is to recommend one *or more* users it can be seen as a multi-label classification problem, which meant that using the sigmoid activation function would likely perform better. However, at this point, none of the activation functions were ruled out.

An initial model like the one just described was implemented. For the recurrent neural network, LSTM units were used. The network was also modelled to only use the titles of posts as input since this was easier than using all of a post's features. The input layer was modelled to let one LSTM unit take one word, from the input sentence, at a time as input. The number of LSTM units was chosen by analysing the data in order to capture a large amount of titles. However, it is still possible to choose any positive number of LSTM units – hence the number of units became a hyperparameter. The words were transformed to indices, where each index corresponds to a row in the embedding matrix, as described in Section 2.6.1.

A problem that quickly revealed itself was the size of the dataset. As the dataset is large, training of the network was very slow even in this simple form. A natural next step was thus to scale down the dataset to allow faster iteration of models. Instead of having a dataset with several thousands of users the dataset was downscaled into only having five users and the posts that they had interacted with. By doing the downscaling, it was faster to train the network, which in turn meant that it was faster to get a result and therefore more experiments could be made. The argument for using this strategy is the assumption that a model performing well on the downscaled dataset will continue to perform well as the dataset is upscaled again.

The goal with this simple artificial neural network model was to establish some starting point to assure that the basic concept was working. To test that the network was learning anything at all we looked for overfitting (see Section 2.5.3). If the model could not overfit on the training data using the downscaled dataset, it would indicate some conceptual error or misinterpretation of the problem. However, once we had verified that the model did overfit, and thus was able to learn from the data, the work moved on to increase the accuracy of the model.

Both the softmax and the logistic activation functions have been used to test the model. With the softmax as the activation function users were picked based on a hyperparameter k, where k was the number of users that would get predicted. This meant that the top k users with the highest probabilities would always get picked as predictions from the model. In practice, this meant that even if only a single user should have been predicted, the model would always predict k users even if the $k-1$ remaining users should not be predicted at all. The reason for using this method to pick users is based on how the softmax function normalises all probabilities. It is not practical to have a condition to pick all users with a probability higher than a certain constant, since the constant will depend on the number of users. This was however supported by the logistic function, which in the end resulted in the softmax function being removed from further testing in the output layer. The limit of when to pick a user or not given a probability introduced a new hyperparameter based on how to set the limit. One option considered was to have a constant limit x, e.g. picking all users with a probability above $x = 30\%$. Choosing x then becomes a problem of optimisation. Another option considered was to pick all users with an above average probability. The hyperparameter then becomes to choose which of the two options to use and to choose a value of x if the first option is used.

It is common to use F_1 score (see Equation 3.3) when evaluating machine learning models [35]. A benefit of using the logistic function over the softmax function was to measure more meaningful performance. The softmax function was not practical because a constant number of users would always get picked when making recommendations. During the performance testing of the network, users were selected based on the probability that the network assigned to them if that probability exceeded a certain threshold. With the more dynamic way to predict users that the logistic function allowed, the recall could be calculated in a more meaningful way which in turn allowed to calculate the F_1 score of the model. With the logistic as the

output function precision, recall and F_1 score was measured and used to evaluate the performance of the model. In order to visualise the performance a TensorFlow tool called *TensorBoard* was used.

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{3.1}$$

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{3.2}$$

$$F_1\ score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3.3}$$

3.3.2 Enhancing the Model

After the performance measures were implemented to calculate precision, recall and F_1 score it became possible to accurately compare models against each other. Because of this, the model could now be extended, and the changes could be evaluated. An enhancement that was easy to implement was regularisation: Both l^2 regularisation and Dropout (see Section 2.5.4) was added to the model. With regularisation two new hyperparameters were introduced; the dropout probability, $P_{dropout}$, and the l^2 factor, β_{l^2}. The options to use either l^2 or Dropout regularisation (or both) also became hyperparameters in themselves.

In order to add more degrees of freedom to the artificial neural network, support for hidden layers was added to the model. The model was implemented so that any non-negative number of hidden layers could be used. For the layers' activation function the ReLU function, described in Section 2.3.2, was used. By adding hidden layers to the model, two new hyperparameters were introduced; the number of hidden layers and how many neurons the layers should have. We simplified the model so that all hidden layers have the same number of neurons. This was to keep the number of hyperparameters, and thus the number of combinations, down to make experimentation easier. Each layer is connected as a chain in a fully connected way, so there are no parallel hidden layers and all nodes between two consequent layers are connected. This means that the output from the recurrent neural network is passed to the first hidden layer and the output from that layer then passed to the second layer and so on – until the output layer. This is illustrated in a simplified way in Figure 3.1.

Another extension of the model was to implement pre-trained word embeddings. The word embeddings would be pre-trained with GloVe [32]. The reason for using word embeddings is that they better capture the relation between words in a language [31], GloVe has also been shown to perform better compared to other methods for creating word embeddings such as word2vec [32]. The first step was to evaluate the performance of the already pre-trained embeddings that can be found at `https://nlp.stanford.edu/projects/glove/`. The two specific embedding matrices that were evaluated are pre-trained on "Wikipedia 2014 + Gigaword 5"- and "Twitter"-datasets. For these embedding matrices, different sizes were tested to find

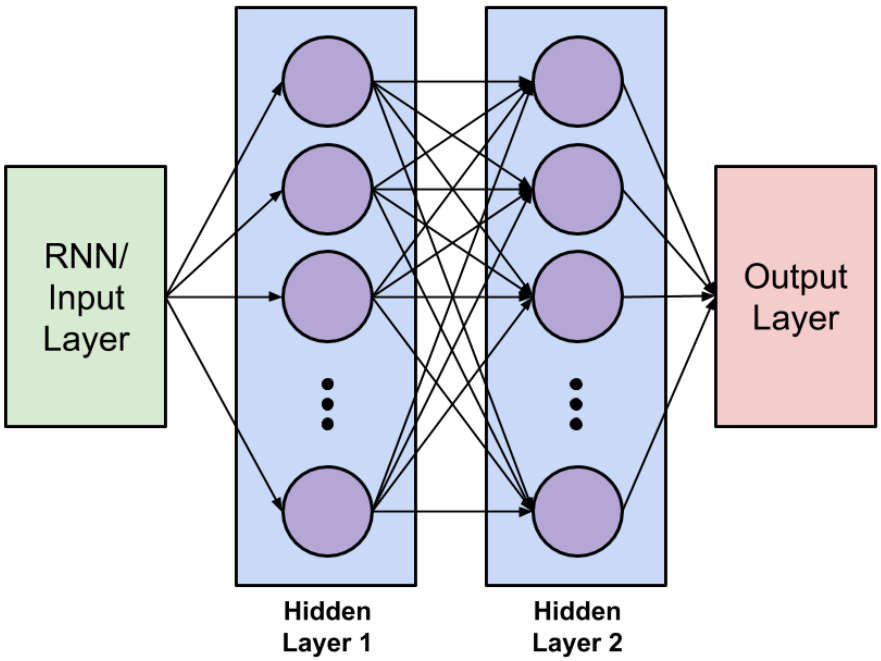


Figure 3.1: A graph visualisation of an artificial neural network with two fully connected hidden layers.

the one that performed best for our task. The second step was to pre-train a new word embedding matrix based on the Reddit dataset. The last step is to compare the accuracy between the embeddings matrices that were trained on "Wikipedia 2014 + Gigaword 5"- and "Twitter"-datasets against the embedding matrix trained on the Reddit dataset. The comparison was completed early in the project and the tests showed clear results of the Reddit's embedding matrix outperforming other embeddings.

The reason for creating new embedding matrices was that some words could be community- or platform-specific, meaning that there was a chance of capturing more relations between the words with this method. Different dimensions of the embedding matrix were evaluated against each other.

To reduce the number of potentially irrelevant details in the data, such as word inflections, a technique called word stemming was used. The goal was to increase the number of shared words between the titles. This could lead to the classifier having an easier time to learn similarities in titles. By applying the Porter stemming algorithm [36], word inflections in the dataset was reduced to their base form. By doing this, the number of unique words is decreased at the same time as word occurrences increase. The potential consequence of using the stemming is that information is lost, information that might be important for making good predictions. This motivates evaluating the network by using both the original and stemmed dataset in the project.

Since there is more data available about each post, such as its content and which subreddit it belongs to, the model was also enhanced to incorporate that. For the subreddits, the actual meaning of the subreddit's name (i.e. the actual text, e.g. *politics* or *humor*) was not of interest. Instead a post's subreddit was encoded as a one-hot vector. This vector was fed through a network layer as a mean to re-size the input's dimensions and was then concatenated directly with the output from the existing recurrent neural network. The option to either use this extra input or not turned into another hyperparameter.

Early in the project subreddits were tested as a classification target instead of users. The network showed good results for predicting which subreddit a post had been posted in. This meant that it could be useful to first train network at predicting subreddits and then use the weights from this network as initial weights when training on users. The reasoning for this was that if the network is already trained to distinguish between posts in a broad term (i.e. in terms of which subreddit they belong to), that could potentially be a good starting point for distinguishing between users' interest. Combining this strategy with using both the title and subreddit of a post as input as previously mentioned was also evaluated.

3.3.3 Tuning Hyperparameters

With the implementation of the model being complete, experimentation and tuning of the introduced hyperparameters was the next step. Hyperparameters are both actual parameters introduced as well as whether to use a certain feature of the model or not. For example, a model with l^2 regularisation turned off is different from one with it turned on – it is not certain that the model will perform better with it turned on. All of the hyperparameters introduced are described in Table 3.3.

As there are a lot of different combinations of hyperparameters, a dynamic and configuration based model was implemented. With this implementation, the whole model can be defined using a single configuration file. The configuration file specifies all of the hyperparameters and when the model was ran, it dynamically built an artificial neural network model to match them. This setup also enabled scheduling the training and logging the progress of different models in an automated way which made experimentation easier. Source code for the artificial neural network model can be found at `https://github.com/recnet/model`.

The goal of tuning the hyperparameters was to maximise the F_1 score for the validation dataset. Under normal circumstances, an optimiser would likely be used to find the optimal parameters that maximised the F_1 score, but since the model is, in fact, an artificial neural network the time it takes for training the model makes an optimiser impractical. Instead, the optimisation relies on systematic testing of different combinations of hyperparameters. Different sets of hyperparameters were randomly generated within some bounds (see table 3.3). This has been proven, empirically as well as theoretically, to give better results instead of doing exhaustive search over all possible combinations. [37]

The typical workflow when optimising the model was thus to look at a previous model that had performed well, identify a hyperparameter that should be changed, test the network with the updated hyperparameter and then compare the results. Which hyperparameter that should be changed could be chosen on a few different basis. For example, with the boolean hyperparameters (i.e. use dropout or not) it is easy to make sure both combinations are tested and if one has not been tested it is a good hyperparameter to pick.

3.3.4 Upscaling

After no further improvements could be made, in terms of F_1 score, on the downscaled dataset due to time constraints, it was time to scale up and evaluate it on a larger dataset with more users. The count of 50 users was selected for the upscaled model.

3.4 Comparing Against a Baseline

When deciding how well a model performs it is compared against a baseline. The baseline puts the accuracy of the model into some context and shows how it measures against other models. In order to have a fair comparison, the users selected are chosen in a similar way as for the artificial neural network model mentioned in Section 3.3.1.

3.4.1 N-grams Based Model as a Baseline

In this project, a model based on N-grams [38] was used as a baseline. The model used was inspired by those in [39], and [40]. Each user is represented by a vector based on the titles they have voted on in the training set. When the model receives a new title, the title is translated into a vector and cosine similarity [41] is used to measure which user the new vector is most similar to. The user whose vector is most similar to the new title's vector is recommended. For more details on implementation, see `https://github.com/recnet/Ngram-baseline`.

3.4.2 Facebook fastText as a Baseline

Facebook has implemented an efficient sentence classifier called *fastText* (`https://github.com/facebookresearch/fastText`). They used the new bag of words techniques mentioned in [42] in order to achieve fast classification while still maintaining accuracy.

The aim of the comparison against fastText is to achieve similar if not better results in terms of accuracy. fastText only allows choosing a constant top k users for the performance measure. In order to compare it to the artificial neural networks model, the best F_1 score was chosen from testing different values of k. Values for k between 1 and n, where n is the number of users in the dataset, were tested. The best F_1 score from fastText is then compared to the F_1 score of the network model. The

performance in terms of speed will not be compared. When running fastText, some parameters can be set to specify e.g. learning rate. These parameters are further explained in fastText's GitHub repository at `https://github.com/facebookresearch/fastText`.

Hyperparameter	Description	Value range
Learning rate	A real valued, positive number scaling the network's gradient during training	[0.05, 0.1, 0.15, 0.2, 0.3, 0.35, 0.4, 0.5]
Batch Size	Positive integer specifying the size of each batch	[25, 50, 100]
RNN units	Positive Integer specifying the number of hidden units in the RNN	[150, 200, 250, 300, 400]
Embedding Size	A positive nteger specifying the size of each word representation in the embedding matrix	[100, 150, 300]
Pre-trained embedding matrix	A binary choice to either use a pre-trained embedding matrix or not	[True, False]
Trainable embedding matrix	A binary choice to either allow or disallow the embedding matrix to be updated during training	[True, False]
Hidden layers	Non-negative Integer specifying the number of hidden layers in the model	[0, 1, 2, 3, 4, 5]
Neurons in hidden layers	Positive integer specifying how many neurons each layer has	[150, 300, 450, 600, 750]
Use l^2 regularisation	A binary choice to either use l^2 regularisation or not	[True, False]
l^2 factor	A real valued scaling factor for the l^2 regularisation	[0.01, 0.05, 0.1]
Dropout regularisation	A binary choice to either use dropout regualarisation or not	[True, False]
Dropout probability	The probability that any given neuron will be kept	[0.5, 0.75, 0.9]
Use constant prediction limit	A binary choice to either use a constant prediction limit over an average based prediction	[True, False]
Constant prediction limit	A threshold such that if it surpassed a recommendation is issued	[0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]
Use subreddit input	A binary choice to either use or not use the given posts subreddit as additional input	[True, False]
Subreddit input neurons	Positive integer specifying how many neurons used in the linear subreddit layer	[50, 100, 200]
pre-train on subreddit	A binary choice to either pretrain the model against subreddits or not	[True, False]

Table 3.3: Table of all hyperparameters present in our model along with a description for each of them

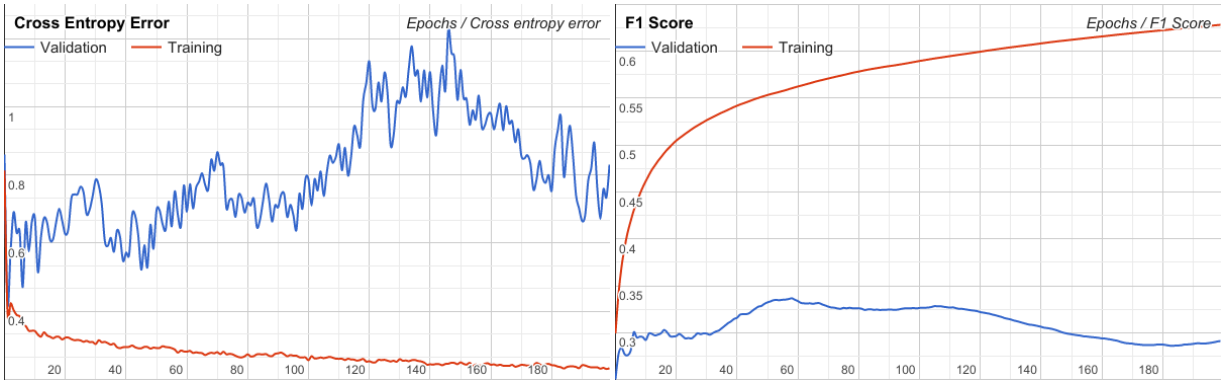
4

Results

This chapter will present the results achieved in the project as described in Chapter 3. These results include comparisons between different models tested and their hyperparameters, but also a look at the different datasets that were used.

4.1 The First Iteration Model

The first iteration of the model was implemented with no hidden layers, as described in Section 3.3. It had a recurrent neural network as input layer with 30 LSTM units. The hyperparameters of this model are found in Table 4.1. When this model was trained on a dataset downscaled to contain five users, overfitting was achieved as shown in Figure 4.1a.



(a) Cross entropy error

(b) F_1 score.

Figure 4.1: The cross entropy error and F_1 score of the first iteration model after 200 training epochs, trained and validated on the training and validation sets respectively. It is good to achieve a low cross entropy error and a high F_1 score.

The best F_1 score achieved on the validation set using the first iteration model was 0.3366, as shown in Figure 4.1b. The precision for this model was 0.2554 and the recall was 0.4936. As seen in Figure 4.1, the model is overfitting for the training data, thus indicating that it is able to learn from it.

Hyperparameter	Value
Learning rate	0.5
Vocabulary size	20000
Batch Size	25
RNN units	30
RNN neurons	200
GRU or LSTM	$LSTM$
Embedding Size	150
Pre-trained embedding matrix	$False$
Trainable embedding matrix	$True$
Hidden layers	0
Use l^2 regularisation	$False$
Use Dropout regularisation	$False$
Use constant prediction limit	$False$
Use subreddit input	$False$
Pre-trained on subreddits	$False$

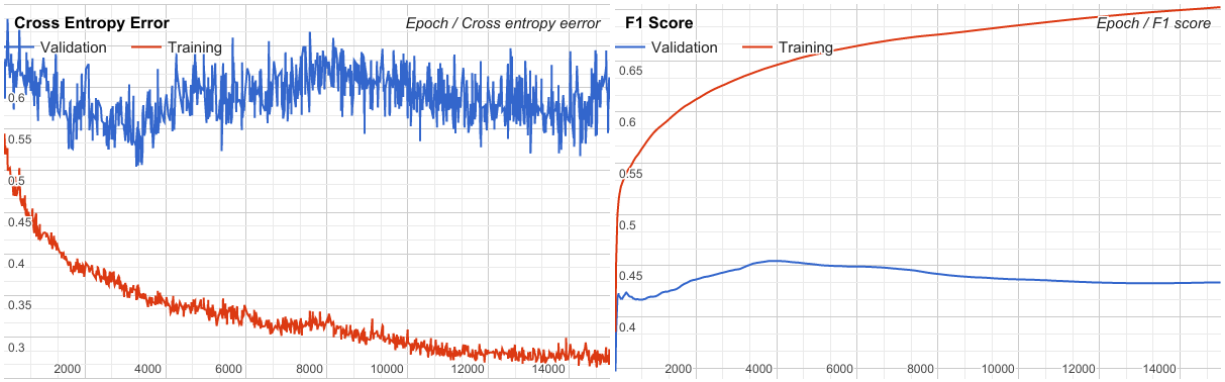
Table 4.1: The hyperparameters for the first iteration model. Non applicable hyperparameters are left out

4.2 The Final Network

After more than 250 different models were tested (see `https://github.com/recnet/data/tree/master/results`), the best performing network we managed to achieve for the word stemmed five users dataset had an F_1 score of about 0.4548 on the validation dataset, as shown in Figure 4.2b. Figure 4.2a shows that the model started overfitting to the training data after 4047 epochs. The precision and recall on the validation dataset was 0.3552 and 0.6368 respectively, for the results on the test dataset see table 4.3. The hyperparameters for this final model are shown in Table 4.2.

When the model was upscaled and trained on the dataset with 50 users an F_1 score of 0.0400 was achieved. The precision and recall are shown in Table 4.3.

As the results from the upscaled model did not scale well, an attempt on the full non-scaled dataset was never done. The model was trained on the word stemmed dataset, the results for this together with the results on the regular downscaled dataset are shown in Table 4.3. The model was trained for 15000 epochs on the dataset with 5 users using an Nvidia GTX 1070 graphics card, which took a little more than 12 hours.



(a) Cross entropy error, achieved on the validation dataset.

(b) F_1 score, achieved on the validation dataset.

Figure 4.2: The cross entropy error and F_1 score of the final model after 15000 training epochs. It is good to achieve a low cross entropy error and a high $F_1 score$.

Hyperparamter	Value
Learning rate	0.05
Vocabulary size	25000
Batch Size	25
RNN units	12
RNN neurons	400
GRU or LSTM	$LSTM$
Embedding Size	300
Pre-trained embedding matrix	$False$
Trainable embedding matrix	$True$
Hidden layers	1
Neurons in hidden layers	300
Use l^2 regularisation	$True$
l^2 factor	0.01
Dropout regularisation	$True$
Dropout probability	0.75
Use constant prediction limit	$False$
Use subreddit input	$False$
Pre-trained on subreddits	$False$

Table 4.2: The hyperparameters for the final network. Non applicable hyperparameters are left out

4.3 Baseline Comparison

This section presents the results of the different baselines tested. These results were used when comparing against the artificial neural network model described in Section 4.2. Discussion about the results are left for Chapter 5.

Final Network Model

	5 users dataset	50 users dataset
	Without stemmed dataset	
F_1 score	0.34	0.0405
Precision	0.273	0.021
Recall	0.5476	0.3699
	With stemmed dataset	
F_1 score	0.3568	0.0400
Precision	0.281	0.021
Recall	0.488	0.3587

Table 4.3: The F_1 score, precision and recall for the final model on the testing dataset. The results for both the regular datasets as well as the word stemmed datasets are shown.

4.3.1 Facebook fastText Classifier

The Facebook fastText classifier was tested on the downscaled datasets for 5 and 50 users and with the corresponding word stemmed datasets. The results are shown in Table 4.4.

Facebook's fastText classifier

	5 users dataset	50 users dataset
	Without stemmed dataset	
F_1 score	0.413	0.0818
Precision	0.414	0.0847
Recall	0.412	0.0791
	With stemmed dataset	
F_1 score	0.557	0.0748
Precision	0.559	0.0774
Recall	0.556	0.0723

Table 4.4: The F_1 score, precision and recall for the Facebook fastText classifier. The results for both the regular dataset as well as the word-stemmed dataset are shown.

The parameters given to fastText to achieve the results in Table 4.4 can be seen in Table 4.5.

4.3.2 N-grams Based Model

The N-grams based model was tested on the downscaled dataset with five users and with the corresponding word stemmed dataset. The results are shown in Table 4.6.
Due to computational complexity a result for the dataset with 50 users could not be achieved within an acceptable time frame for this baseline.

Parameter	Value
lr	0.1
lrUpdateRate	100
dim	10
ws	5
epoch	5000
minCount	1
minCountLabel	0
neg	5
wordNgram	2
loss	ns
bucket	10000000
minn	0
maxn	0
thread	8
t	0.00001
pretrainedVectors	[]

Table 4.5: Command line parameters used to get the recorded results for fastText. Documentation for each of the parameters can be found at `https://github.com/facebookresearch/fastText`

N-grams based model		
	5 users dataset	**50 users dataset**
	Without stemmed dataset	
F_1 score	0.4068	–
Precision	0.2903	–
Recall	0.6793	–
	With stemmed dataset	
F_1 score	0.4180	–
Precision	0.2998	–
Recall	0.6903	–

Table 4.6: The F_1 score, precision and recall for the N-grams based model. The results for both the regular dataset as well as the word-stemmed dataset are shown.

4.4 Analysis of the Datasets

This section presents the result of the investigation into the Reddit dataset that was used. The goal of this was to see if there were any irregularities or patterns in the dataset that could have had an affect on the performance of the networks or the baselines.

4.4.1 Downscaled Dataset with 5 Users

It was found in the training set for the downscaled 5 user dataset that the difference in length of the longest title and the second longest title was large. The longest title was 1928 words while the second longest was 66 words. This skewed the variance for the training data and is why the standard deviation is 24.9, as seen in Table 4.7. If the outlier is removed, the standard deviation is brought down to 9.7 which is similar to the validation and testing sets.

Feature	Training	Validation	Testing
Titles			
Amount	6927	1983	992
Median length	9	9	8
Average length	12	11	11
σ title length	24.9	9.6	9.4
Longer than average	2198	696	320
More than 1 vote	76	9	6
More than 2 votes	2	0	0
Subreddits			
Amount	103	65	59
Average voted in	32.2	21.8	19.4
σ voted in	18.4	11.49	10.2

Table 4.7: Statistics regarding title length, votes and subreddits for the downscaled dataset containing five users. σ is the standard deviation.

The analysis also showed that few users shared liked titles. In the training set only about 1% of titles have more than 1 vote – in the validation and testing sets that number is even less, as shown in Table 4.7.

The vote density for the downscaled 5 user dataset is shown in Table 4.8. The table shows statistics for the training, validation and testing sets. It summarises how many subreddits a user has voted in, the total number of votes a user has made and how many votes a user has made in the two subreddits that the user has voted most in.

User	Subreddits	Total votes	Votes in two most voted subreddits
Training data set			
Ayavaron	64	1411	444
akkartik	14	1387	1333
crmaki	14	1410	902
izzycat	37	1363	505
doctorgonzo	32	1429	586
Validation data set			
Ayavaron	40	416	133
akkartik	8	392	372
crmaki	11	390	252
izzycat	26	428	158
doctorgonzo	24	364	146
Testing data set			
Ayavaron	33	172	60
akkartik	5	220	212
crmaki	12	194	131
izzycat	19	207	77
doctorgonzo	28	204	78

Table 4.8: Vote distribution showing votes in subreddits, total votes and votes in the top two voted subreddits for each user in the training, validation and testing data sets with five users

4.4.2 Downscaled Dataset with 50 Users

Compared to the downscaled dataset with five users there are no big outliers in the downscaled dataset with 50 users. There is however a higher percentage of users that share liked titles in comparison. As seen in Table 4.9, for the training set, 8.59% of titles have more than one vote. In the validation set it is 6.65% and for the testing set it is 6.2%.

Feature	Training	Validation	Testing
Titles			
Amount	63176	18486	9301
Median length	9	9	9
Average length	12	12	12
σ title length	11.7	9.3	9.3
Longer than average	23474	6736	3354
More than 1 vote	5428	1231	577
More than 2 votes	980	220	92
Subreddits			
Amount	343	248	192
Average voted in in	31.28	22.6	18.58
σ voted in	17.6	12.17	9.88

Table 4.9: Statistics regarding title length, votes and subreddits for the downscaled dataset containing 50 users. σ is the standard deviation.

The statistics regarding the vote density, similar to the one in Table 4.8, for the 50 users datasets have been left out simply due to the space that the table would require. However, most of the Reddit posts in the 50 user dataset have a low number of votes.

5
Discussion

This chapter will discuss the obtained results from Chapter 4 and potential problems with the methodology leading to them.

5.1 The Models and Their Performance on the Datasets

The best performing network, as seen in Section 4.2, got an F_1 score of 0.3568 on the five users dataset. F_1 score is a combination of precision and recall. The precision for this network was 0.281 and the recall 0.488, which means that out of all recommended users about 78% would not be interested, and about 51% of the users that should get predicted were not. As one of the motivating factors for this thesis was to lessen the abundance of uninteresting notifications that users of social media get, the low precision can be seen as a poor result as it more often results in irrelevant notifications than not. However, the low density of user votes, as seen in Table 4.8, for a given post could mean that a lot of users have not seen a post and therefore not reacted to it. That they have not seen a post could still mean that the post is of interest to them, but that interest is not reflected in the dataset. A user study would potentially capture the interests of the users better than the dataset and therefore give a more accurate performance measure than just checking against a validation set. An example of this can be found in [43].

It is interesting to investigate the hyperparameters that were used and what conclusions we can draw from them. First of all, as seen in Table 4.2, the best performing network did not use any pre-training or additional data (i.e subreddit as input). This could indicate that our assumption, that subreddits could be used to predict a user, was wrong. When looking at the top five performing networks that we tested for both the 5 and 50 users datasets, it seems that the choice of unit types in the recurrent neural network (i.e. *LSTM* or *GRU*) did not matter. Looking at the same top networks, we can see that almost all of them use the dynamic prediction limit (predicting all users with an above average certainty). This is most likely because the ones that should be recommended have a higher probability and vice versa, the prediction boundary then cuts between these two groups in an efficient way.

To test our assumption that hyperparameters that were tuned for a smaller amount of users would scale to more users, the model that performed best on the dataset with 5 users was also tested on the dataset with 50 users. The result of upscaling the number of users to predict but keeping the hyperparameters the same was

disappointing. The new network performed poorly on the dataset that it was not designed for, as seen in Table 4.3. This suggests that it will not be possible to use a smaller network (in terms of users) in order to solve the problem with more users. As a consequence, it will be hard to keep updating the model since a real application gets more users daily. The fact that it does not scale well is not that surprising in hindsight. More labels mean a greater complexity, so it is not certain that the degree of freedom in the model optimised for five users is enough, at the same time as a larger degree of freedom in the model for five users means that it overfits to the training data too quickly.

Another assumption that was mentioned in 3.1 was if a user has voted on a post that indicates that the user found the post enjoyable or interesting, and a lack of interaction means that the user did not find it interesting. However, if a user has downvoted a post, that could potentially mean one of two things; either the user did not like the content and does not want to see similar posts again or the user finds the post interesting but does not agree with the point of view of the author. Similarly, with no interaction, a user might simply not have seen the post, has not liked it, or has not cared enough to vote on it. There is no clear way of knowing what is wrong or right without talking to a large subset of the users.

When comparing our model to the baselines, the network is consistently worse than fastText and the n-grams based model(as can be seen in Section 4.3). These results were consistent regardless of whether the dataset was word stemmed or not. In its current state, with the model's F_1 score being comparable to that of the N-grams model and not better than that of Facebook's fastText, it is hard to motivate its use considering the time it takes to train. There are several reasons that could potentially explain why the network did worse than fastText and a few of them are discussed in Section 5.2. One of these reasons could be that fastText only recommends the one user it is most certain of, while our model recommends all users over a certain threshold thus introducing a margin of error.

5.2 The Datasets' Influence on the Final Results

An interesting discovery made when examining the datasets was the scarcity of posts with more than one vote, especially in the downscaled five users dataset. As mentioned in Section 4.4.1, only about 1% of titles in the training set for 5 users have more than one user that has voted on that title – that is, only 1% of the titles have overlapping users, and it is even less for the validation and testing sets. Having that few titles with user overlap could mean that the best way of choosing which users to recommend is to always pick the one or two most certain users. Another possibility seeing as there is little overlap between subreddits is to go for a purely statistical approach; given an input title and subreddit recommend the user that is most active in that subreddit. The problem with both of these approaches is that they are not very practical, in practice, there is most likely more than one or two people that would like to get a recommendation seeing as how popular Reddit posts can get tens of thousands of votes.

Another consequence of votes being distributed in the way just described is that recommending more than one user is most often wrong, as any model that recommends two or more users, in that case, will get at least one of them wrong. This will make it difficult to evaluate a model for a real world usage, which presumably does not share a similar distribution since the network should most certainly want to recommend more than one user for that case. Most users have very few votes in the majority of all subreddits they have voted in. This shows that users tend to be most active in just a small subset of subreddits. An example of this is the user *akkartik*; in Table 4.8 we can see that of the total 1387 votes, 1333 of these are from two subreddits. Even though not all users are as devoted to a particular subreddit, there is a trend of keeping to a few subreddits and being mainly active in those. The fact that a user almost exclusively visits one, or a small number of, subreddits could result in a filter bubble, as described in [44]. A filter bubble occurs when a user only likes a few similar categories of data, a system that recommends new content will then naturally only recommend content from those categories. The aim of a recommender system is to recommend content that a user would not otherwise find, to then only recommend content from the same category is therefore not so useful as the user would likely find it anyway. Avoiding a filter problem is a complex task that was not the purpose of this project. It is still interesting to consider though, and some of the work proposed in Chapter 6 could potentially help.

Another interesting result was that the network trained on the 5 users dataset benefited from the word stemming while the network trained on the 50 users dataset did not. As stated in [45] stemming only helps in cases where we have sparseness in the data.

6
Future Work

During the work with this thesis a lot of different hyperparameters and methods were tested, but as time was a serious constraint, there are still some interesting areas left untested. This chapter discusses areas to focus on in future work based on this thesis.

6.1 Improving the Model Architecture

A problem with the current choice of model is that the maximum number of users that it can predict without having to be re-trained is fixed. This makes this solution unfeasible in a commercial application with a growing number of users. One way to counter the problem of a fixed number of users would be to train one network per user. This solution should work just as well as the model presented in this thesis as a network that models a function that takes some input of length n and outputs a result of length k. This is equivalent to k networks with input of length n and output of length 1 [46].

A constraint introduced in order to simplify hyperparameter searching was to limit the number of neurons in the hidden layers to be the same for each layer. This might not be the best way of modelling the networks. Many networks instead follow the geometric pyramid rule, which suggests that the number of neurons should shrink with each layer [47]. However, as some of the best performing networks tested had only zero to one hidden layer, it is not obvious that this extension would help – unless the reason for the worse performance of the models with more hidden layers is because the geometric pyramid rule was not followed. Nonetheless, it could be worth examining.

Another constraint for simplification is that the ReLU function is exclusively used as activation function in the hidden layers. A problem with the ReLU function, however, is that something called dead neurons can occur. This is common when the ReLU function is used as it evaluates to zero when the input is less than or equal to zero. There is, however, a way to reduce the number of dead neurons with something called Leaky ReLU. Leaky ReLU is another function that uses a small negative slope instead of just being zero for negative input [48]. A proposal is to try Leaky ReLU or other activation functions as alternatives to just using ReLU.

6.2 Re-modeling Input and Output

If work with the Reddit dataset is continued, there are a few things to consider. Firstly, a potential improvement for the input to the network is to use a separate recurrent neural network for the subreddits when using both titles and subreddits as input. This assumes that the name of a subreddit will have an impact on the model performance. The reasoning is that some subreddits have similar names (e.g *machinelearning* and *learnmachinelearning*) or have similar semantic meaning. This motivates using a recurrent neural network with character-based vectors, as opposed to word-based vectors for the title input.

Additional to the title and the subreddit, there are more features to the Reddit data that have not been used. One example of this is the actual content of a post. When creating a post on Reddit, it must have a title and must belong to a specific subreddit, beside this the author can also choose to add some additional (optional) content. This extra content could be text, images, or hyperlinks. The intuition behind using this extra information is that there might be some pattern between what type of content a post has and a user's interest in that post. There is a drawback with using the content of posts however; some analysis on the dataset revealed that only a small percentage of the posts had any extra content at all – the rest only had a title and a subreddit.

The problem with the model having a maximum number of users it can recommend, as described briefly in Section 6.1, could also be solved by re-modeling the network's output. Instead of recommending individual users the network could potentially be used to recommend more general *user types* or *user profiles*. User profiles could be constructed from features of the users, and the model would then try to recommend which user profiles would be interested in a certain post.

7
Conclusion

The model proposed in this thesis performs comparably to already existing solutions such as Facebook's fastText and N-grams based model. Because of the computational complexity of using an artificial neural network and the long training time it does not seem like a good approach to the problem. There are some potential solutions that could make the model better as proposed in Chapter 6, but in its current state, the model is not so good in real world usage. In the real world scenario of serving recommendations on a large social network, the economical cost of running such a complex model would likely be too great – especially as the complexity increases as more users are introduced.

Considering the low precision, even on the smallest dataset tested, it is also unlikely that the model proposed would solve the problem of increased stress described in Chapter 1 – rather it would probably contribute to the problem in its current state.

We also learned that the dataset used might not be optimal for solving the problem of recommending users in the way examined in this thesis. It is not clear though if it is a problem with this particular dataset or if there is an underlying problem with datasets of this kind (i.e. social platform user behaviours).

We can conclude that the best model proposed in this thesis has no real advantages over other similar methods, but there is room for improvement (like described in Chapter 6).

References

[1] Slack Technologies Inc. (2017). Highlight word notifications, [Online]. Available: https://get.slack.help/hc/en-us/articles/201398467-Highlight-word-notifications.

[2] T. Westermann, S. Möller, and I. Wechsung, "Assessing the relationship between technical affinity, stress and notifications on smartphones", in *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*, ser. MobileHCI '15, Copenhagen, Denmark: ACM, 2015, pp. 652–659, ISBN: 978-1-4503-3653-6. DOI: 10.1145/2786567.2793684. [Online]. Available: http://doi.acm.org/10.1145/2786567.2793684.

[3] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering", *IEEE Computer Society*, 2003.

[4] I. Basheer and M. Hajmeer, "Artificial neural networks: Fundamentals, computing, design, and application", *Journal of microbiological methods*, vol. 43, no. 1, pp. 3–31, 2000.

[5] R. Lippmann, "An introduction to computing with neural nets", *IEEE Assp magazine*, vol. 4, no. 2, pp. 4–22, 1987.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, pp. 1–28, http://www.deeplearningbook.org.

[7] D. Michie, D. J. Spiegelhalter, and C. Taylor, *Machine Learning, Neural and Statistical Classification*. 1994, pp. 1–5.

[8] I. Rish, "An empirical study of the naive bayes classifier", in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, IBM New York, vol. 3, 2001, pp. 41–46.

[9] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers", in *Proceedings of the fifth annual workshop on Computational learning theory*, ACM, 1992, pp. 144–152.

[10] C. Cortes and V. Vapnik, "Support-vector networks", *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[11] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology", *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[12] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch", *The Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, 2011.

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[14] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.

[15] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks.”, in *Aistats*, vol. 15, 2011, p. 275.

[16] R. A. Dunne and N. A. Campbell, “On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function”, in *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne, 181*, vol. 185, 1997.

[17] P. Golik, P. Doetsch, and H. Ney, “Cross-entropy vs. squared error training: A theoretical and experimental comparison.”, in *Interspeech*, 2013, pp. 1756–1760.

[18] C. M. Bishop, *Pattern recognition and Machine Learning*. 2006, vol. 128, p. 206.

[19] G. E. Nasr, E. Badr, and C. Joun, “Cross entropy error function in neural networks: Forecasting gasoline demand.”, in *FLAIRS Conference*, 2002, pp. 381–384.

[20] C. M. Bishop, *Pattern recognition and Machine Learning*. 2006, vol. 128, pp. 209–210.

[21] TensorFlow, *Python api r1.1*, Apr. 2017. [Online]. Available: `https://www.tensorflow.org/api_docs/python/tf/nn/sigmoid_cross_entropy_with_logits`.

[22] O. Shamir, “Making gradient descent optimal for strongly convex stochastic optimization”, *CoRR*, vol. abs/1109.5647, 2011. [Online]. Available: `http://arxiv.org/abs/1109.5647`.

[23] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization”, *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: `http://arxiv.org/abs/1412.6980`.

[24] 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.

[25] B. Neyshabur, R. Tomioka, and N. Srebro, “Norm-based capacity control in neural networks.”, in *COLT*, 2015, pp. 1376–1401.

[26] H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. Ward, “Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval”, *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 24, no. 4, pp. 694–707, 2016.

[27] S. Hochreiter and J. Schmidhuber, “Long short-term memory”, *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[28] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions”, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[29] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling”, *ArXiv preprint arXiv:1412.3555*, 2014.

[30] J. Turian, L. Ratinov, and Y. Bengio, “Word representations: A simple and general method for semi-supervised learning”, in *Proceedings of the 48th annual meeting of the association for computational linguistics*, Association for Computational Linguistics, 2010, pp. 384–394.

[31] T. Mikolov, W.-t. Yih, and G. Zweig, “Linguistic regularities in continuous space word representations.”, in *Hlt-naacl*, vol. 13, 2013, pp. 746–751.

[32] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation.”, in *EMNLP*, vol. 14, 2014, pp. 1532–1543.

[33] R. Lowe, N. Pow, I. Serban, and J. Pineau, “The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems”, *ArXiv preprint arXiv:1506.08909*, 2015.

[34] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *Tensorflow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `http://tensorflow.org/`.

[35] Y. Yang and X. Liu, “A re-examination of text categorization methods”, in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, ACM, 1999, pp. 42–49.

[36] M. F. Porter, “An algorithm for suffix stripping”, *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[37] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization”, *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[38] W. B. Cavnar, J. M. Trenkle, *et al.*, “N-gram-based text categorization”, *Ann Arbor MI*, vol. 48113, no. 2, pp. 161–175, 1994.

[39] Y. Miao, V. Kešelj, and E. Milios, “Document clustering using character n-grams: A comparative evaluation with term-based and word-based clustering”, in *Proceedings of the 14th ACM international conference on Information and knowledge management*, ACM, 2005, pp. 357–358.

[40] A. Khabia and M. Chandak, “A cluster based approach with n-grams at word level”, *International Journal of Computer Applications*, vol. 117, 2015.

[41] M. Steinbach, G. Karypis, V. Kumar, *et al.*, “A comparison of document clustering techniques”, in *KDD workshop on text mining*, Boston, vol. 400, 2000, pp. 525–526.

[42] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, “Bag of tricks for efficient text classification”, *ArXiv preprint arXiv:1607.01759*, 2016.

[43] J. H. P. Suorra and O. Mogren, “Assisting discussion forum users using deep recurrent neural networks”, in *Proceedings of the 1st Workshop on Representation Learning for NLP at ACL*, 2016, p. 53.

[44] E. Pariser, *The filter bubble: What the Internet is hiding from you*. Penguin UK, 2011.

[45] P. R. Christopher D. Manning and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[46] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”, *Neural networks*, vol. 6, no. 6, pp. 861–867, 1993.
[47] T. Masters, *Practical neural network recipes in C++*. Morgan Kaufmann, 1993.
[48] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models”, in *Proc. ICML*, vol. 30, 2013.

A

The Reddit dataset

The Reddit dataset was retrieved from `https://www.reddit.com/r/redditdev/comments/bubhl/csv_dump_of_reddit_voting_data/`. The dataset was published by the user *ketralnis* who is an employed administrator on Reddit. The data comes directly from the official Reddit databases.

The raw data as retrieved contained three fields per data point which represent a user, a post ID and a vote. Every data point in the dataset indicates that a specified user has either up- or down-voted a certain post on Reddit. An excerpt from the dataset looks like shown in Table A.1.

Username	Post ID	Vote
	⋮	
2bornot2b	t3_899vr	1
2bornot2b	t3_89as2	-1
2bornot2b	t3_89az7	1
2bornot2b	t3_89b84	1
2bornot2b	t3_89c4k	1
2bornot2b	t3_89de8	-1
2bornot2b	t3_89e1e	1
	⋮	

Table A.1: An excerpt from the raw dataset with the field username, post ID and vote.

This raw dataset contains $7,405,561$ votes for a total of $31,927$ users and $2,046,401$ posts. The dataset only contains votes from users who have actively agreed to make their votes public. The dataset contains up to the latest 1000 up- and down-votes (for a maximum of 2000 votes) for each user. The action to either upvote or downvote a post on Reddit is used on the website to rank posts against eachother where an upvote indicates that a user likes a post and a downvote indicates the opposite.

Figure A.1: An example of a post on Reddit and how it can be either up- or down-voted. Screenshot from `http://reddit.com/` on April 4, 2017.

The posts on Reddit are written by users on the platform. A post always has a title, and it always belongs to a so called *subreddit*. The post seen in Figure A.1 has the title *Colorized by me: Abraham Lincoln, June 3, 1860* and it belongs to the subreddit *pics*. A subreddit can be seen as a category – there are many subreddits on Reddit and posts belonging to the same subreddit are usually similar.