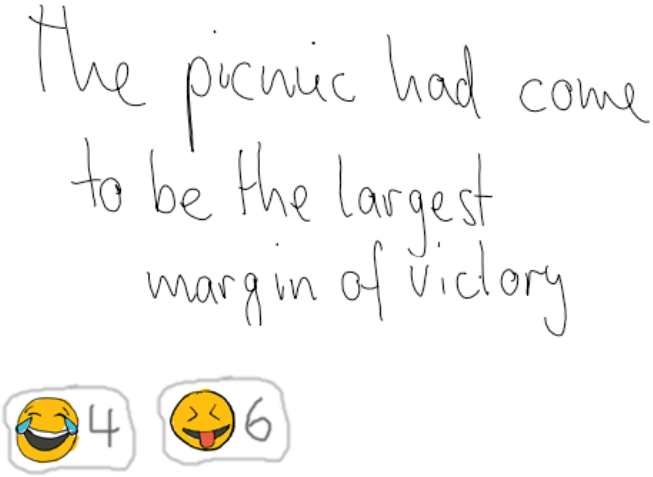

Information retrieval in instant messaging platforms using Recurrent Neural Networks

Assisting Discussion Forum Users Using Deep Learning

Master's thesis in Algorithms Languages and Logic

Jacob Hagstedt Persson Suorra

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016

Information retrieval in instant messaging platforms using Recurrent Neural Networks

Assisting Discussion Forum Users Using Deep Learning

Jacob Hagstedt Persson Suorra

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Computing Science Division
The Machine Learning Research Group
Chalmers University of Technology
Gothenburg, Sweden 2016

Information retrieval in Instant Messaging platforms using Recurrent Neural Networks
Assisting Discussion Forum Users Using Deep Learning
JACOB HAGSTEDT PERSSON SUORRA

Supervisor: Olof Mogren, Department of Computer Science and Engineering
Examiner: Peter Damaschke, Department of Computer Science and Engineering

Master's Thesis 2016
Department of Computer Science and Engineering
Computing Science Division
The Machine Learning Research Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Sentence sampled by the LSTM Network during training, primed with "the". The sentence was then handwritten using Graves recurrent neural network handwriting generation demo[1]

Typeset in LaTeX
Gothenburg, Sweden 2016

[1] `http://www.cs.toronto.edu/~graves/handwriting.cgi`

Information retrieval in instant messaging platforms using Recurrent Neural Networks
Assisting Discussion Forum Users Using Deep Learning
JACOB HAGSTEDT PERSSON SUORRA
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

We present a discussion forum assistant based on deep recurrent neural networks (RNNs). The assistant is trained to perform three different tasks when faced with a question from a user. Firstly, to recommend related posts. Secondly, to recommend other users that might be able to help. Thirdly, it recommends other channels in the forum where people may discuss related topics. Our recurrent forum assistant is evaluated experimentally by prediction accuracy for the end–to–end trainable parts, as well as by performing an end-user study. We conclude that the model generalizes well, and is helpful for the users.

Keywords: Deep learning, Recurrent Neural Networds, RNN, natural language processing, NLP, Language model, Machine Learning, Recommendation model, Instant Messaging, Slack.

Acknowledgements

I would like to start of by giving a huge thanks to my academic supervisor, Olof Mogren. Without him, this thesis would never have become a reality. Furthermore, I would like to thank both of my supervisors at Netlight. I also want to thank my friend Emanuel Snelleman and dear cousin David Ebbevi for the hours of brainstorming together. Lastly I would like to thank Netlight for the opportunity to do my thesis at their Stockholm office and the ability to test this thesis idea using their Slack data.

Jacob Hagstedt Persson Suorra, Stockholm 2016-05-03

Contents

List of Figures

List of Tables

1
Introduction

There are many instant messaging (IM) platforms on the market right now. Google Hangout, Facebook Messenger, KakaoTalk, WhatsApp and Slack, just to name a few. All of the mentioned services are used by millions of users and often one user utilizes many instant messaging tools. The use of multiple communication systems not only partitions all of our data, but it also increases the risks of losing it. When we do not know from which service the data was sent or who sent the data, it becomes harder to find it and in that way, the quality of the data decreases.

All kind of discussion forums pose an interesting setting for human interaction. Chat systems, as mentioned previously, social media, and customer support systems are closely related. These platforms play an increasingly important role for people, both in their professional and personal lives. As an example, many software developers are familiar with web services such as Stack Overflow where you can ask questions and get answers from other users. These situations are also present in customer support systems, allowing for quick turnaround times and a growing database of queries that can be made available to customers, along with their responses.

There are many companies that want to solve one part of this puzzle, namely the partition of information. Slack is the company announcing themselves as the company making *Team communication for the 21st century*[1]. They want to change the way teams communicate by moving them away from emails and into a more instant messaging type of communication. They want to gather everybody that is using some sort of IM tool into the same platform, preventing loss of data and knowledge[2]. This might have solved the problem of organizing an organization's data, and communicating within a large team. However, this is likely to increase the amount of data. Making it easier to share files, communicate with a whole team at the same time and utilizing channels for specific topics will encourage users to produce more data.

In this thesis we will present an automated system that can help people make better use of existing platforms, and we propose a system that solves some of the associated problems. In this thesis we will focus on the instant messaging tool Slack.

[1] `https://slack.com/is`
[2] `https://slackhq.com/we-made-a-video-41229c874985#.q2i3v748e`

1.1 Neural networks

Inspired by biological neural networks, such as the human central nervous system, artificial neural networks are a machine learning technique, a sub-field of computer science, with intentions of learning autonomously [3]. One of the first algorithms used for neural networks were described in 1943 by Warren McCulloch and Walter Pitts. They paved the way for other algorithms and methods such as unsupervised learning with their McCulloch Pitts Neuron [14]. Together with these neurons it were possible to build models that resembled a biological brain, connected into a network called a perceptron [4]. The perceptron is a self-organizing or adaptive system proposed by Rosenblatt. With the original intention of the word "perceptron" as a generic name for a variety of theoretical neural networks, it has become the definition of an algorithm for supervised learning of binary classifiers [20][6].

However, the development of neural network algorithms stagnated in 1969 when two major problems were encountered: The output of an XOR circuit could not be resolved using a single layer perceptron and the current computing power available was not enough for the long training times required by the neural networks [16]. According to Stanford Natural Language Processing Group, training deep architectures of neural networks before 2006 was unsuccessful [4]. What has changed now is the development of new methods for unsupervised pre-training, more efficient parameter estimation methods, together with a better understanding of model regularization. Accurate models for regularization is very important in order to prevent over-fitting. This is the problem where the network adapts too much to the training set, which could have systematic noise. This is something that is not acceptable and will make a neural network perform bad.

Deep neural networks, as in neural networks with multiple layers, has recently shown much promise for Natural Language Processing (NLP) with very promising applications. Using neural networks to utilize context in order to solve problems has now become possible thanks to recent advancements. For example due to fast GPU-based [5] implementations, the development of recurrent neural networks [5] and also deep feedforward neural networks. Neural networks' ability to learn to represent context is a big benefit when using them in areas with a lot of natural language, for example together with instant messaging services. This will be the focus of this thesis.

1.2 Motivation

With more conversations, resulting in more data, information overflow is becoming a big concern. As the amount of data becomes even larger, one could claim the

[3] http://global.britannica.com/technology/machine-learning

[4] http://nlp.stanford.edu/projects/DeepLearningInNaturalLanguageProcessing.shtml

[5] http://www.kurzweilai.net/how-bio-inspired-deep-learning-keeps-winning-competitions

quality of the data decreases as it takes even longer time to find one specific part of data. Finding connections takes more time and seeing the bigger picture inside a team's communication is becoming impossible. If it could be possible to let a service analyse the input into Slack, and from that insight understand what is being written in order to present the users with related topics, conversations and problems, faster turnaround times from a question to an answer could be achieved.

Handcrafting connections between multiple topics is time-consuming and might even be impossible. We are not quite sure what to find, so letting a computer do this for us is very compelling. Since the raw data is humans talking to other humans, a computer ideally have to understand, in some sense, natural language. It is with this motivation this thesis will utilize a deep recurrent neural network to get a better understanding of all conversations for a certain company using Slack.

Training a deep neural network to actively analyze the data could possibly increase the quality of the data, since information could be more accessible and problems will hopefully not be solved twice. A deep recurrent neural network could also give a very much improved search functionality for Slack. Today Slack provides a quite advanced search functionality where we can search across multiple channels and even inside shared files, but it is still the case that one searches on a word-by-word basis. If it could be possible to utilize a deep recurrent neural network, more results could be provided, since a user does not need to phrase herself in a particular way; the network might be able to find related information anyway.

1.3 Data integrity

Gathering information is not so easy and straightforward as it might sound. This is especially true if the information we are gathering is not something like a production line's productivity data (such as speed, number of errors per hour etc). When it becomes information about people and what they are talking about, what they are doing etc., one needs to think one step further about how this data is handled.

When it comes to personal data on the web, or anywhere it is accessible to someone else, it is important to keep in mind what is gathered and why. If we gather some information which is not needed to improve the service proposed in this thesis, and the collected information is somewhat private, then we want to skip collecting this. This venture between collecting certain data or not in order to provide enough data for the service to work with is difficult. In the context of this thesis, the service would like to have as much data as possible, in order to provide the best results. But we also does not want to risk exposing people's privacy. Planning and discussing this before publishing the service can help.

Data privacy must also be viewed of from multiple angles. Some privacy issues are sourced by law, others by economical interests (such as trade secrets), yet others by personal preferences. In order to cover as much of these points as possible, and still be able to get as much data as possible, a decision was made to only use the

company's public channel Slack data. This means that we will not use any data from direct messages between users, nor data from private channels. As a result of this decision, the data used is the same data that all users of the team can already access.

1.3.1 Wanting to be forgotten

An initial approach when dealing with data collection or data analyzing, when private or personal information might be present, is to clearly state what is needed for an algorithm or service and why it is so. Maybe some personal information can rightfully be gathered, as long as the affected person is notified and prompted to opt-in, rather than opt-out. Also letting the private data used by the service only be visible for that particular person could help. An example of this is the web browser Google Chrome, which given your own browsing history can autocomplete certain URLs in order to make your experience better. What is important in this case, and many others, is also a person's ability to be forgotten if he or she wants to stop using a service.

This problem of wanting to be forgotten was what Google experienced about 1.5 years ago, when the European court of Justice ruled in favor of Mario Costeja González, a Spaniard which wanted to be forgotten by one of Google search's search results [6]. One of the points made during this trial was that

> *"data-processing systems are designed to serve man; ... they must, whatever the nationality or residence of natural persons, respect their fundamental rights and freedoms, notably the right to privacy, and contribute to ... the well-being of individuals;"*

Another point to consider is the question of who is owning the data that is used by services. In Sweden, a person does not 'per default' get full rights to data generated by herself. If a stick figure is drawn, the person will not get copyright for that drawing per default, since someone else could have drawn that stick figure beforehand. A person does, however, get copyright automatically under the Copyright Act if the data reaches up to the requirement of "originality". In that case, the Copyright law applies to all types of information. Other than that, a person can also actively seek protection under the Patent Act. In this case there are demands on ingenuity.

The applicable laws of copyright changes if the data is generated as a part of a job. "Work made for hire" comes in place when the work generates data by an employee as part of his or her job. In this case an agreement can grant the company the person is working for the full rights to the data created. In this case, gathering information and data, the data ownage will fall to the company. The company could then, in practice, do what they want with it. This however, does not mean that they should. If the information includes private information about you it becomes sensitive and

[6] `http://curia.europa.eu/juris/document/document.jsf?text=&docid=152065&amppageIndex=0&doclang=en&mode=lst&dir=&occ=first&part=1&cid=276332`

handling it carelessly is not desired.

In this thesis, the service will need to be able to inform the employees about what it gatherers, why and how it will be used is an important task which should be done as soon as possible. This comes back to the point made earlier; it might be better to let people opt-in rather than opt-out. The problem with this might be that the amount of data available can be significantly decreased and the resulting service may not perform as well as desired. It is also appropriate for the service to only be directly accessed within the already established network at the company where the service will be running.

1.4 Goals

The goals of this thesis is to design a system which in real-time analyses a user's input and suggests three types of recommendations. Firstly, the system will suggest related posts. Secondly, the system will recommend users that might be able to help. Thirdly, the system will recommend other channels in the forum where users may discuss related topics. In order to do this, we must have a way of extracting what a user is inputting into the communication tool in use, namely Slack.

The service must also be able to analyze the input and have a way to connect back to the user, showing the results.

In more detail, the system must be able to parse natural language, in particular, what needs to be decided is:

- When does a user post a question?
- How can we find related posts?
- How do we return results to a user?
- How can the service know if the results returned are accepted by the user?

This system should work in real-time, meaning in contrast to a non-real-time system that is running in batch mode, it should not gather all the users conversations at a specific time each day and give users recommendations the day after, but this should be done as the user is typing his or her message into Slack.

It should also be possible to make requests directly to the system, asking it for information such as

- What kind of conversations is talked about the most?
- Given a person, what is his/her most talked about topic?

1.5 Approach

The approach for this thesis will be split into two major parts:

1. The first part will be about finding posts given other posts, which are most similar. This will in turn be split into two parts

   (a) Using pre-trained word vectors to get a vector representation of a Slack post. These vectors will then be used to calculate similarity using cosine similarity.
   (b) Using a LSTM network, pre-trained as a language model, which will hopefully yield a better representation of the posts.
2. The second part is closely related to the first part. Here the system will, in addition of finding posts which are most similar to other posts, find users and forum channels that are best related to the content of the post. This can be due to the fact that said user often talks about the post's content or is very active in a channel which the post's content is talked about.

1.5.1 Part one

The first part will be about finding similar posts. This will in turn be split into two parts: using pre-trained word vectors and using the internal state of an LSTM network.

The pre-trained word vectors will be used to fast get a demo up and running, in order to see if the hypothesis, on which the method of finding similar posts rely upon, works. It will also serve as a good benchmark in order to see if the results are improved when later replacing them with an LSTM network.

1.5.2 Part two

Using what has been learned in part one, in addition to finding posts, finding users is a small but very interesting step. Here a LSTM network will be used to recommend users who write similar posts and forum channels where similar posts are discussed. The idea is that this can be used by users to know who to talk to when they experience certain problems, and to be informed about channels where the question's topic is often discussed.

In order to use a network to recommend users and channels, the network will not be trained to predict the next word, given a history of words, but instead predict a user and channel. This will be done by letting the network get Slack posts as input and the user who created that post, together with the channel where the post was made, as targeted outputs.

1.6 Limitations

The core focus of this thesis is to utilize the results of deep neural networks, and their ability to hold context in order to find more relevant information. With that, linguistic ideas to interpret language will not be a part of this thesis. Also any deeper research for finding out best practises for tokenizing words, filtering out stop words or any other more advanced pre processing will not be a part of the thesis. Instead we will rely on the LSTM network to do most of this kind of work. Mainly because of the lack of time, but also because LSTMs have shown much promise

when trained as a language model, to capture the essence of the content.

Other limitations is that not much time will be put into finding what is the best way to inform the users about the findings. One possibility is to use a Slack-bot to simply reply to the user with the results. The user could then reply with a Slack reaction if they thought the result given was good or bad.

1.7 Achievements

The overall results shows that a recurrent neural network is able to retrieve relevant information from an instant messaging platform. The latest progress in natural language processing and sequence based LSTM networks have become powerful enough to process natural language to such a degree that they can be useful in production. With this thesis, a proof-of-concept with a virtual assistant has shown much promising results in handling multiple users at once, and the results given by the network was satisfactory for the users.

Even more positive results were shown in part 2 of the thesis, where a network was trained in a more end-to-end fashion where it, given a Slack post, returned user and channel recommendations. These results proved to work really well, and the overall impression from the users were positive.

1.8 Related work

Interest in deep neural networks to process natural language has increased the latest years. LSTM based recurrent neural networks, in particular, are very well suited for long sequences and has shown very promising results when used as a language model to process natural language[24]. Training networks to read natural language remains a challenge, but has shown very promising results the latest years, especially in regard to QA-systems [17].

Recurrent neural network based language models (RNNLM) has shown state-of-the-art performance when used in a variety of tasks. In the paper by Mikolov T and Zweig G, they improve the performance of the RNNLM by providing a contextual real-valued input vector in association with each word. Using this, they achieved state-of-the-art results using the Penn Treebank dataset [15]. Other related work is Alex Graves' RNN for generating sequences [11]. There he showed how LSTM networks can be used to generate complex sequences by only predicting one data point in time. His network was trained using 2 dimensional coordinates, in order to generate handwritten text with very nice results. This required the network to keep track of long sequences in order to predict next point.

Sequence-to-sequence models are often used for machine translation. For example the paper by Sutskever et.al. Here they present a general end-to-end approach that makes minimal assumptions on the sequences structures and are still able to achieve

state-of-the-art results on an English to French translation task using the WMT-14 dataset [24].

Keyword extraction, and its related topic information extraction, using LSTM based networks are in the progress, and shows right now promising results. The paper on optimization of neural networks in the context of keyword search by Gandhe et.al. haven't used a recurrent network, but instead a feed-forward neural network based language model in order to improve keyword search in natural language [7].

As already mentioned, language models based on recurrent neural networks have shown promising results, especially compared to a more traditional statistically way of predicting next word in sequences, namely n-grams. The paper Neural Probabilistic Language Models [2] has researched the problems of dimensionality; that a word sequence on which the model will be tested is likely to be different from all the word sequences seen during training. One product that is in use today, that relates to this on a concrete level is SwiftKey Neural[7].

SwiftKey is a company that delivers a keyboard app for mobile devices. Their app have been downloaded over 250 million times [8]. The fall of 2015 SwiftKey released an alpha of their latest keyboard: SwiftKey Neural. It utilizes a deep neural network to predict the next word by making use of a neural network's ability to understand context. Where the old N-gram technology would predict the sentence "It might take a " with the words [look | few | while], SwiftKey neural instead comes back with [while | little | few]. Here the word "while" makes more sense as the top suggested word instead of "look". SwiftKey has this to say about their latest innovation:

> *"N-gram technology provides accurate predictions for common phrases and those learned from you. However, it has some limitations, as it can't capture the underlying meaning of words and can only accurately predict words that have been seen before in the same word sequence"*[9]

Kumar et.al behind the Ask Me Anything [13] paper claims that most tasks can be casted into question answering (QA) problems over language input when it comes to natural language processing. Using their Dynamic Memory Network they was able to achieve a mean accuracy of 93.6%, compared to 93.3% accomplished by Facebook's MemNN.

[7] `https://blog.swiftkey.com/neural-networks-a-meaningful-leap-for-mobile-typing`
[8] `https://swiftkey.com/en/keyboard/android`
[9] `https://blog.swiftkey.com/neural-networks-a-meaningful-leap-for-mobile-typing/`

2
Theory

In this chapter the basic related theory will be addressed and their corresponding terms and expressions used by Artificial Neural Networks. In particular, the terms used when training and running recurrent neural networks will be explained.

2.1 Recurrent Neural Networks

The recommendation system used in this thesis is based on deep recurrent neural networks (RNNs) and will be used to solve the three different problems proposed in Section 1.4.

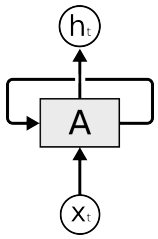

Figure 2.1: A recurrent neural network have loops[1]

A recurrent neural network is a type of neural network that, unlike a more vanilla feed-forward network, not only connects all of their neural units forwards to the next layer, but also connects back to themselves within a layer. This can be seen in Figure 2.1, where the input x_t going into the RNN cell not only produces an output, h_t, to connect to the next cell or output, but also has a feedback loop. This gives RNNs an internal state which can be utilized when using the network for sequential input, for example a sentence.

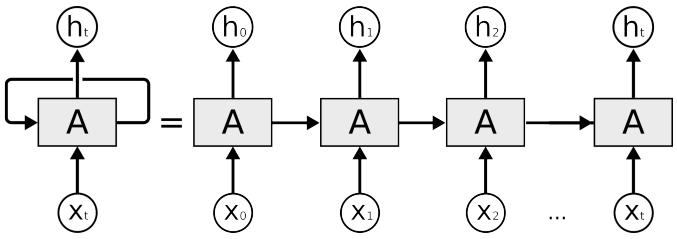

Figure 2.2: An unrolled recurrent neural network[1]

The sequences can more easily be visualised if the RNN is unrolled into its different time steps. This can be seen in Figure 2.2, where the RNN is unrolled t times. If

[1] `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

the sequenced used by an RNN would be word based, then each word in a sentence would be fed into each input x_t, in order. RNNs has shown very promising results the recent years, used for hand recognition [10], character based RNNs for predicting next character [2], learning simple context-free and context-sensitive languages [9] just to name a few. The RNN, however, suffers from two problems: The vanishing and exploding gradient problem.

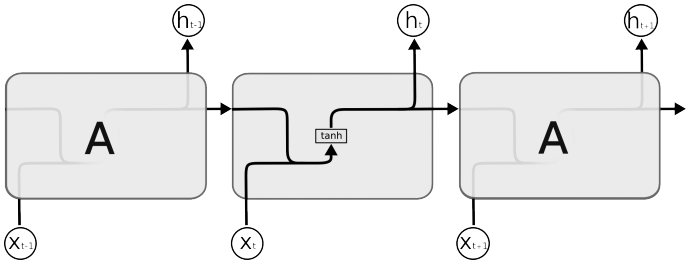

Figure 2.3: The repeating module in a standard RNN contains a single layer[3]

When a recurrent neural network is trained, its internals are updated using gradient decent. This means that the network will take a small step in the direction where the cost decreases. The vanishing gradient problem is the problem where information is lost over time when using long sequences. The vanishing gradient problem is related to both the values of the weights, as well as the derivatives of the cell's activation function. Figure 2.3 shows the internals of an RNN cell as well as its *tanh* activation function. Due to the chain-rule, which is used to calculate the gradient, each cell's gradient can be calculated individually, which will make the error signal propagate through the entire network [3]. The derivative of the tanh-function will be smaller than 1 for all inputs, except 0. This means that unrolling an RNN multiple time-steps will lead to a vanishing error signal. All this means that RNN will not perform as good as one would hope when using longer discrete sequences than 5-10 time steps between corresponding inputs and targets [8], since the information is lost on the way.

The presence of the vanishing gradient problem is fully dependent on the magnitude of a cell's internal error signal. If the magnitude is larger than 1, the network will experience the precise opposite - the exploding gradient problem. Both of these problems are not easy to handle for the network, since it will in each step either increase or decrease the values exponentially. Studies have shown that it is the vanishing gradient problem that typically happens, making the RNNs hard to train. This is one of the reasons the Long Short-Term Memory (LSTM) cells was introduced in 1997 [12].

[2] `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`
[3] `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

2.1.1 LSTM cells

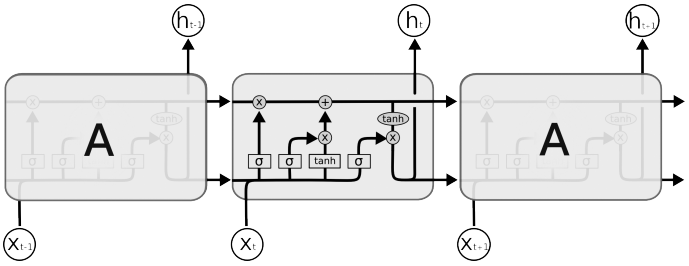

Figure 2.4: The repeating module in an LSTM contains four interacting layers [3]

LSTM networks does not suffer from the vanishing gradient problem (but still the exploding). The error signal in an LSTM cell is added to the cell's memory, resulting in even the smallest error signal to be fed back to the next time step, and will not decrease in an exponential fashion. This can be seen in Figure 2.4 where the top-most horizontal arrow is the LSTM cell's internal memory. This makes LSTM networks very good at keeping track of even the longest sequences and have been used successfully for language modelling (predicting the distribution of the word following after a given sequence), sentiment analysis [25], textual entailment [19], and machine translation [24].

2.2 Language model

A Language Model's (LM) purpose is to learn the structure of a language. More specifically, it is a statistical model which is used to learn the joint probability function of sequences of words in a language. Due to the dimensionality of words, this is not an easy task. For example, if a model should give the joint distribution of 10 consecutive words in a natural language with a vocabulary of size 100,000, there are potentially $100000^{10} - 1 = 10^{50} - 1$ free parameters. Up until the beginning of the 21st century this has been modelled using n-gram approaches, which has been very successful to get a good generalization of the language [1]. In the referenced paper, however, an approach using neural networks is demonstrated and since then RNN based LMs (RNNLM) has been trained to predict the next word, given a history of sequences, very successfully.

2.2.1 Softmax

A Softmax classifier is build upon the binary Logistic Regression classifier, but is generalized to multiple classes. The binary Logistic Regression classifier can be used when there is only two possible outcomes: dead or alive, win or lose, etc. whereas a Softmax classifier is used when there are multiple classes.

$$\sigma(\boldsymbol{z})_j = \frac{e^{\boldsymbol{z}_j}}{\sum_k e^{\boldsymbol{z}_k}} \tag{2.1}$$

The Softmax classifier function can be seen in Equation 2.1, and works by giving a probability real value to each class. It is mainly used to produce a probability distribution over all the classes a network can classify.

2.3 Training

In order to train a RNNLM there are multiple parts needed: A way to extract the next word given a sequence of input words, a cost function which can be either maximized or minimized and a way to decide how to change the RNNs internal weights, just to name a few. Here we will touch upon four of the needed parts: Softmax layer, Cross entropy loss function, epoch and backpropagation

2.3.1 Cross entropy

$$cross(p, q) = -\sum_{k} q_k * \log p_k \tag{2.2}$$

In order to train a neural network, an optimization function is needed. This function is often expressed as a cost function, making the problem an optimization problem where the cost function should be minimized. There are several different cost functions used when training neural networks. One of the most common are Classification Error or Mean Squared Error. When training recurrent neural networks, however, the Cross entropy loss function is often preferred when the output is a probability distribution, such as a Softmax.

It works by comparing the network's prediction with the desired target. Equation 2.2 shows how two vectors, p and q are compared where p is the predicted output and q is the desired target. The measure is based on two different probability distributions over the same set, and when used in recurrent neural networks it gives a good measurement of how closely the calculated probability distribution reflects the target probability distribution.

During training of a neural network the term epoch is often used. It is defined as a complete pass through of all the training data used [4]

2.3.2 Backpropagation

Training of the model used in this thesis will be done using backpropagation through time (BPTT) and minibatch stochastic gradient descent. The principle behind backpropagation is to calculate the gradient of the loss function (See Section 2.3.1) against the weights in the network [21]. The gradient is then fed into the loss function, which in this case is gradient descent, which updates all the weights in the network.

[4] `https://www.cse.unsw.edu.au/~billw/mldict.html#epoch`

2.4 Preventing overfitting

Here, some regularization techniques will be addressed, as well as other techniques which are used to prevent a neural network to overfit the data set. One of the biggest problems faced when using neural networks are when the model overfits your training data and is unable to capture the characteristics of the data set.

Overfitting is the problem where a network has been trained too much on the sampled training data set, making it bad at generalizing. This leads to the network performing bad in the case that new data was to be sampled. It also means that it will perform very good when feeding the network with the training data, but it will fail to grasp the overall picture, and will perform bad when giving it new, never before seen, data.

2.4.1 Dropout

To prevent overfitting, a number of methods can be used. One of the newest methods, which has proven to work very well on feed-forward networks is dropout [22]. When talking about neural networks, dropout refers to the action where some nodes in the network is reset to a zero or random state during training. It is often executed by randomly selecting which nodes to reset, and randomly selecting when to do it.

Unfortunately, it has also shown to work rather bad when using RNN or LSTMs due to their recurrent nature, so either some other methods need to be used, or a well performed technique for applying dropout on RNN needs to be used. One way of still using dropout on RNNs are to only apply it on a certain subset of the RNNs' connections [28].

2.4.2 Network Size

Other ways to regularize the network is to increase the size of the data set or to change the size of the actual network. When using this approach to prevent overfitting it is really the proportions between the amount of training data available and the size of the network that is important. If the network is just big enough to provide an adequate fit given the amount of data, it will be difficult for it to overfit. This means that if it is possible to increase the amount of training data, that could be a good approach to decrease the risk of overfitting. If that is not possible, decreasing the network size by decreasing the number of units (in this thesis' case, the number of LSTM units) used.

2.4.3 Early Stopping

The last way of regulating a network is to use two sets of data when training: one training and one validation data set. The idea is then to only train the network using the training set and every now and then run the network using the validation data set in order to see how well it performs on data it has not trained to know, but

has the same characteristics. This can be done multiple times during the training. The idea is then that the error for the training data set will decrease, and if the validation error starts to increase from a previous value, the training of the network is stopped [26].

2.5 Pre-processing

In order to get the most out of the available data, pre-processing it before training a network is a good approach for achieving better results faster. It is one of the most difficult parts when handling much data and can therefore be very time-consuming [18].

There are different types of pre-processing techniques available when handling natural language. A typical strategy when using English as language, is to use Punkt Tokenize[5]. For this thesis, a lot of pre-processing will not be performed, since one of the desires of this thesis is to let the network do as much work as possible and to do as little manual work as possible.

2.5.1 Vocabulary

Vocabulary is the word used to describe all the words the LM is able to recognize. It is the vocabulary that decides the output size of the Softmax layer used by the RNNLM (See Section 2.2.1). If a RNNLM has a vocabulary of 10 words, the output size of the Softmax layer needs to be of size 10, in order to give each word a probability.

2.5.2 Tokenization

Tokenization, in the context of lexical analysis, is the process where a sentence of words are broken into a list of tokens. This list of tokens is the input for further pre-processing, or in the case of this thesis, the input to the RNNLM used. One of the reason to split text into tokens is to minimize the size of the vocabulary, which makes it easier for the model to generalize. For example, instead of having to save *we're*, *you're*, *we* and *you* as words in the vocabulary, only *you*, *we* and *'re* needs to be saved.

2.5.3 Cut-off parameter

This is one of many parameters used for this thesis. This particular parameter is closely related to the size of the vocabulary, since it directly decides the size of it. The cut-off parameter decides how frequent a word must be across all the data used for training, in order to belong to the vocabulary set. By not including words that only occur once or twice, a lot of words can be removed from the vocabulary. If a word is not relatively frequent, then it will most probably not be important for the

[5] `http://nlp.stanford.edu/software/segmenter.shtml`

RNNLM to know in order to generalize about the language used.

Filtering out the unusual words will also most probably filter out misspellings. Having a small vocabulary but still being able to cover a majority of the text used for training will speed up the training, and make the network converge faster. This will also help to make the model fit on the GPU memory, used for training.

2.6 Cosine Similarity

$$sim(r_1, r_2) = \frac{r_1 \cdot r_2}{\|r_1\|\|r_2\|} \tag{2.3}$$

Cosine similarity is a measurement used in vector algebra. It is a used to calculate the similarities between two vectors. Since $cos(0) = 1$, two vectors that have an angle between them of zero degrees will get a similarity of 1, whereas two opposite pointed vectors will have a cosine similarity of -1. The cosine similarity function is shown in Equation 2.3

3
Method

This section describes how the process will be carried out during this thesis. In particular, how the training data will be created and the pre-processing will be done. Here will also follow a short description of all the special tokens that will be used, especially the Slack specific notations that needs to be replaced for the tokenizer to work well.

This section will also mention a more in-depth explanation of how the vocabulary will created, as well as the idea of how to find similar posts, users and channels needed for step 1 and 2 in this thesis (See Section 1.4).

3.1 Training data

The networks that will be created and trained during this thesis will all be pre-trained as a language model and will focus on the English language. Therefore the LSTM network are in need of a lot of natural language texts in order to do a proper training.

The training will be done over several epochs. This means that the network will be trained several times on the same training data, each time changing the networks' underlying weights in order to learn to generalize about the training data.

As of 2016-04-14, the company's total Slack data consists of 1.47 million messages. This could seem as a lot, and might be enough to do proper training. The problem is that of these messages, only 15% are from public channels. The rest of the messages are direct messages or in private channels [1]. Due to privacy reasons, the direct messages and private channels will not be used for this work, as described in Section 1.3. This results in around 226.1k messages that can be used for training.

3.1.1 Pre-training

Due to the low number of Slack posts that can be used, a pre-training will be done on the complete English Wikipedia, in order to prime the model with generic English language. For this, the complete dump from 2015-03-15 was used[2]. Wikipedia is particularly good as a training dataset to prime a LM, since it contains a lot of

[1] `https://my.slack.com/admin/stats`
[2] `https://dumps.wikimedia.org/`

sentences and articles that cover a lot of different topics.

The Wikipedia dataset will be cleaned using the Wiki-Extractor[3] before continued pre-processing. The network will then be trained one epoch using the Wikipedia data set. After this epoch of training, the network will have seen a vast amount of different sentences, and will have learned the basic structure of the English language. After the Wikipedia pre-training, the training will continue using the Slack data. Here the network will continue to train, just as it did using the Wikipedia dataset, but instead use the Slack data as training data.

The idea behind using two datasets for training is to first let the network learn to generalize on the English language as a whole, and then to specialize on the language and phrases used by the company's users on Slack.

3.2 Pre-processing

In order to get the most out of the available data, pre-processing the data before training a network is a good approach for achieving better results faster. Pre-processing is one of the most difficult parts when handling much data and can therefore be very time-consuming [18].

There are different types of preprocessing techniques available when handling natural language. A typical strategy when using English as language, is to use Punkt Tokenization[4]. For this thesis, a thorough pre-processing will not be performed, since one of the desires of this thesis is to let the network learn, and to do as little manual preparation work as possible. For example stemming, which is the method of counting inflected forms of a word together, will not be performed.

3.2.1 Wikipedia

The preprocessing of Wikipedia articles will be done in 5 steps described below. Wikipedia provides a monthly dump of all available articles on its site. It can be found on Wikimedia's download page. The one used for this project will be the English Wikipedia[5].

1. Use the Wiki-Extractor, which is an Open Source project for extracting and cleaning of a Wikipedia database dump. It outputs into a number of files of similar size over multiple directories[3]. This makes it more convenient when parsing all the articles, instead of handling a bigger, single file

2. For each directory created, load all the files within that directory and tokenize all numbers across all articles. This results in all numbers having the same single, predefined, token. Also, all text is made lower case in order to not count

[3] `https://github.com/bwbaugh/wikipedia-extractor`
[4] `http://nlp.stanford.edu/software/segmenter.shtml`
[5] The 20150315 dump: `https://dumps.wikimedia.org/`

a word twice, just because it is present with a capital letter. The files are then concatenated into a long list of articles for convenience, then compressed and saved to disk. Now, instead of having multiple directories containing multiple files, there are one file per directory previously created by the Wikipedia Extractor.

3. Each file in step 2 contains a list of articles. For each article, tokenize using the NLTK *word_tokenizer* method [6] with the Punkt Tokenizer[7]. The result will be a list of tokens, instead of a string. These new lists are saved in a list on disk.

4. Step 3 has created a matrix, where each row is a tokenized article. This step is to create a counter. This counter will, for each article, count the occurrence of all words. This will in the end be a counter that maps each occurring word in all articles to the number of occurrences. This is saved on disk.

5. The last step is to create a set of words which will become the Wikipedia vocabulary (See Section 2.5.1). This is done by loading the counter created in step 4 and filtering out all words that does not occur the number of times decided by the cut-off parameter (See Section 2.5.3). This is then saved to disk.

3.2.2 Slack

Preprocessing the Slack data follows similar steps as for the Wikipedia data. The Slack data can be obtained using the Slack Export page [8]. This will export all message history in public channels, links to files shared in public channels, archived public channels, a list of all users as well as an activity log for integrations. What is not included in the export is data from private channels, direct messages and edits and deletion of logs.

Unlike the Wikipedia dump, the Slack export will also include information about all the users and channels available on the forum. This is parsed too, together with the actual posts created on Slack.

The processing of Slack posts are done in three major steps, where the second step includes a number of substeps. In contrast to when processing the Wikipedia data, the Slack data is much smaller, making it unnecessary to split each substep into actual steps and saving the progress each time.

[6] `http://www.nltk.org/api/nltk.tokenize.html#nltk.tokenize.punkt.PunktLanguageVars.word_tokenize`

[7] `http://www.nltk.org/nltk_data/`

[8] `https://my.slack.com/services/export`

3.2.3 Slack tokens

The Slack data consists of some special tokens, which was not faced when processing the Wikipedia data. Slack uses some special tokens to identify different Slack-specific notations. Some of the special notations include the @-token. This token is used by users to notify other users within a channel. The @-token can be one of the following three:

- *@channel* to notify all members of a channel
- *@here* to notify all members that are online. This will, in contrast to the *@channel* token, only notify those who have a Slack client session active at the moment
- *@username* to notify a specific user

These tokens are not presented in the exported Slack data as *@<token>*, but uses some special series of characters. For example, the *@here* token is, in the exported Slack data, represented as *<!here|@here>*.

More special texts include code snippets, which are represented by either surrounding text with one grave accent: `` `<code here>` ``, or three: ` ```<code here>``` `. These are often used when monospace font is wanted, and is therefore very handy when sharing code over Slack.

When links such as URLs are shared on Slack, they are wrapped by angle brackets: `<link>`.

The last type of special tokens used by Slack that will be handled are Emojis. The Emojis used by Slack are a spin on common emoticons [9]. Slack also offers the ability to add custom Emojis. Common for all Emojis are a unique name. This name can only contain lowercase letters, numbers, dashes and underscores. The Emojis are noted in text by surrounding colons. An example of a simple smiling Emoji is `:smile:`. Also special for Emojis are the Emoji categories. These are how the Emojis are categorized. For example, the previous Emoji example belongs in the "people" category, whereas an Emoji of a boar belongs to the "nature" category etc.[10].

Since we will use NLTK's Punkt Tokenizer to tokenize all text, it is vital to make all these special Slack tokens into one single character, otherwise they will be split up into several tokens. This would make it harder for the network to learn when to add these special Slack tokens, since, for example, colon can be used elsewhere in text, without necessarily being an Emoji.

[9] `https://get.slack.help/hc/en-us/articles/202931348-Emoji-and-emoticons`
[10] `http://www.emoji-cheat-sheet.com/`

3.2.4 Slack Steps

Here follows the steps that will be used to pre-process the Slack data before being used for training the network:

1. The first step is part of an "importing" step, where all the exported Slack data is imported into a local database. Here all users, channels as well as all posts for each channel are imported and stored in a database for more convenient and faster access.
2. All the posts stored in the local database are loaded. Here follows the substeps done
   (a) Make all text lowercase. This is done in order to make the counting of word occurrences case-insensitive.
   (b) Just as with the Wikipedia data, replace all numbers with a number token
   (c) Replace code snippets with a code token.
   (d) Replace all @-Slack-tokens with a specific at-token, one for each type.
   (e) The exported Slack-data uses the XML's versions of less-than and greater-than signs, > and < respectively. Replace these with normal < and > signs.
   (f) Replace all Emojis with a token representing the Emoji's category
   (g) Lastly, for this step, is to replace all links (Http etc) with a link token. This is done since we do not want the network to try to learn specific links, but only when to use links.
3. This steps includes both step 3 and step 4 from the Wikipedia processing steps. The text for each post are tokenized using the NLTK's Punkt Tokenizer and a counter is created to count the occurrence of all words used. Just as with the Wikipedia data, the list of tokens created by a Slack post is appended to a list, which together with the counter is saved to disk.

3.3 Vocabulary

In order to do the pre-training on the Wikipedia data, as well as continuing with the Slack-data, both of the two vocabularies created by the two datasets will be combined into the final vocabulary that will be used. They will be added as two sets, ensuring that words that occur in both vocabularies only will occur once in the final vocabulary. This will hopefully make a better vocabulary to use, since the input data is the company's Slack conversations. We will get a lot of useful words from the Wikipedia data set, as well as IT specific words from the Slack data set. These are words that, when looking at only the Wikipedia data set, may not be so common, but are very useful for an IT consultant company. Also added to the vocabulary are all the unique tokens created to identify all the Slack specific tokens.

3.3.1 Training data

In order to use the data for training, we need to process it by replacing all the tokens not present in the final vocabulary with a special UNK-token (for unknown). Both

the Wikipedia data used for pre-training, as well as the Slack posts needs to be processed.

3.4 Finding Slack posts

The first part of this thesis is to find similar Slack posts, given a Slack post. This will be done using two approaches: using pre-trained word vectors and using the internal state of a trained LSTM network.

Both of the implementations will give a vector representation of a Slack post. The vectors are constructed a bit different, and the details of the two implementations will be described in the following sections. When the vectors are constructed they will be compared using cosine similarity (See Section 2.6). Once the network is built and trained the new representation will hopefully give us more relevant posts.

3.4.1 Pre-trained word vectors

Pre-trained word vectors will be used for two reasons. Firstly, they will be used in order to setup a demo in order to see if the hypothesis, on which the method of finding similar posts rely upon, works. It will also serve as a good benchmark in order to see if the results are improved when later replacing them with the internal state of an LSTM network.

When using the pre-trained word vectors, the idea is to get a representation of a Slack post by summing all the included words in the post's vectors (filtering out things like smilies, links etc).

Using pre-trained word vectors also speeds up this step, since it will not be a big part of the project. This step will lead to two major results: a good and fast way of getting started working with the Slack data, and to get an idea of how working with word vectors work. This step will also give the already mentioned base line to use for comparing the two approaches.

One of the flaws with using pre-trained word vectors and summing all included words in the way that will be done, is that a long post with a lot of text where a lot of the words are just stop words, can shift the vector into a wrong direction. These words will make the post longer without adding any interesting information that can be used to find other posts.

"I was wondering if there is anyone here that could help me find some information about best practices when it comes to Python code documentation?"

vs

"What are the best practices on Python code documentation?"

Figure 3.1: Example 1

Figure 3.1 shows the principle where a long text will not necessarily add any more useful information in regard to using pre-trained word vectors. Taking the first sentence and filtering out english stopwords yields the sentence:

"wondering find information practices Python code documentation?"

Doing the same for the second sentence yields:

"practices Python code documentation?"

Figure 3.2: Example 2

So by having posts with a lot of stop words might skew the final vector into a direction that might not capture the important information. This is where the LSTM will come into power. By utilizing an LSTM's ability to learn and remember what is important, longer posts will be more similar to shorter posts, as long as they talk about the same thing(s).

3.4.2 LSTM Language Model

The idea for the LSTM is to use the internal state already present in an LSTM network (See Figure 2.4), and use that as a representation of a post. The idea is that this representation better handles context and remembers what is important in a text, compared to allowing all words to be weighed the same.

The LSTM will be build using Tensorflow[11], a new Open Source library for Python for Machine Intelligence. Both networks used in this thesis will be deep RNNs, based on a language model. The goal of this step is to fit a probabilistic model that can assign probabilities to words. The models are inspired by the results obtained from Zaremba et al., 2014[28], which achieves very good results on the PTB dataset[12].

3.5 LSTM Training

The training of the LSTM network will be very similar both when using the Wikipedia data and when using the Slack data. The training will be fairly straightforward. For each epoch, the trainer will start of by sampling a sentence, 10 words long, primed with "the". This will be done in order to empirically check the network while it's

[11] `https://www.tensorflow.org/`
[12] `https://www.cis.upenn.edu/~treebank/`

training. This lets us read the sentences sampled and see if they becomes more clear or not. After sampling, the actual training begins. After each training epoch, the network is sampled again.

3.5.1 Training a Language Model

The training of a Language Model is done by feeding the network with both an input and a target vector. The input data is a sentence, which has been tokenized into a list of tokens, described in Section 3.2.1 and Section 3.2.2. The target is a copy of the input vector, but it is time-shifted one step. One token in a sentence counts as one step. Since the idea behind the language model is to predict the next token, given a history of tokens, each input step needs to be mapped to the next token in a sentence during the training. In order to speed up the training process, the training will be done in batches, where the network will be fed 20 sentences in each iteration.

	a)	b)	
It's raining cats nad dogs!	[It, 's, raining, 〉 cats, <unk>, 〉 dogs, !]	[8, 236, 992, 110, 45823, 〉 22311]	[236, 992, 110, 45823, 22311, 702]

Figure 3.3: Showing one example of training data that is fetched by the network in each iteration. (The actual length of the sentence in this example is not representative for the lengths that will be used during this thesis' training) **a)** shows the first 6 tokens of the original sentence. **b)** shows 6 tokens, time-shifted one step

For each iteration, during one epoch, new data is fetched from the training data and input and target matrices are calculated. Figure 3.3 shows an example where the first column shows a sentence with one misspelled word. The second column shows how a sentence is first tokenized, together with replacing all the unknown words. The last two steps involves converting the tokens into numerical form, by changing all tokens into a unique ID.

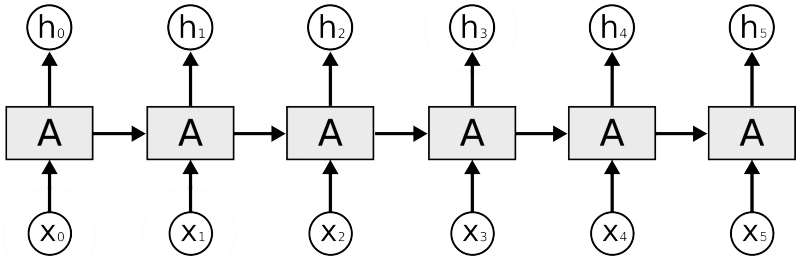

Figure 3.4: An unrolled RNN, unrolled 6 times[13]

The training then continues by feeding each ID number from Figure 3.3, step 3a) into each input x_t, shown in Figure 3.4 as input and feeding each ID number from Figure 3.3, step 3b), as targets at h_t. The training will then use cross-entropy loss (See Section 2.3.1) to train the network's internal weights and biases.

[13] https://colah.github.io/posts/2015-08-Understanding-LSTMs/

3.5.2 Training a Recommendation model

For the second part of this thesis, a model that is a continuation of a language model will be trained in a end-to-end fashion to recommend forum users and channels. This model will be called Recommendation model.

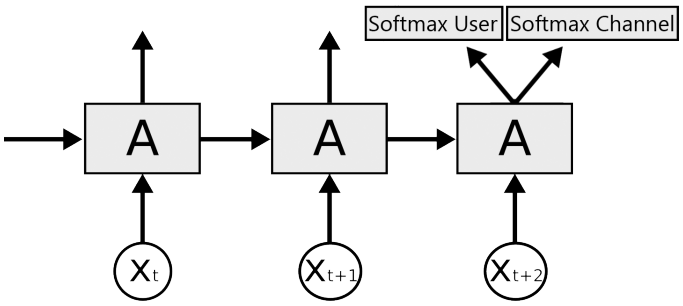

Figure 3.5: The layout of our Recommendation model.[14]

Training of the Recommendation model is quite similar as to train a language model. This model will have two softmax layers (See Section 2.2.1), one for each type of recommendation: users and channels. Figure 3.5 shows the layout of our Recommendation model, with the two softmax layers attached to the end of a deep recurrent LSTM network modelling the input. These are softmax layers with the number of outputs corresponding to the number of users and the size of channels in the forum, respectively.

Like the training of the language model used in step 1 of this thesis, the Recommendation model will be pre-trained using the Wikipedia and Slack data sets, after which it will be trained in an end-to-end fashion to recommend users and channel. The input will be similar as with the language model, where all the Slack posts will be fed. The targets for the two softmax layers, users and channels, will be the Slack post's creator and the channel in which the post was made.

In order to get better results, data augmentation will be performed[27] on the Slack data, where some posts will be duplicated as well as modified. Since there exists channels with very few members that still produce a lot of data, and also channels with all employees that produce far less data, the ratio between the number of posts and number of members in a channel will be calculated and inspected. If a post is created in a channel where the ratio falls below a decided value, there will be a 20% chance that the post will be duplicated as a response to the low ratio. Furthermore, there will be a 50% for the duplicated post to have its tokens changed. This will be done in order to create some noise and let the network converge faster and prevent overfitting. Also, some filtering of the number of users and channels will be made. Inactive users (having authored less than 10 posts) and channels with fewer than 50 posts will be removed.

[14]Figure inspired by and based on `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Since the Recommendation model will be pre-trained as a language model, the idea is to use the model to recommend all three different recommendations.

3.6 Evaluation

In order to know if using a more advanced approach, by training an LSTM network, is a good way to find the most relevant posts, evaluations of the network's results will be performed. Evaluations will be done in multiple steps. Firstly, a manual comparison of the results given when using the pre-trained word vectors versus the results given by the LSTM network will be done. Secondly, a user satisfaction study will be performed, evaluating actual performance of the service in a live setting in the live system with users interacting with it. The assistant will encourage users to provide feedback on the recommendations by giving the posts made by the service reactions, which we will interpret as positive or negative. If more than 60 minutes goes without a reaction, we count this as one "No reaction". This will mean that a recommendation can receive multiple positive and negative reactions, but only one "No reaction". For the post recommendations, the answers will be served either by the LSTM state representation, or by the pre-trained word vector representation, randomly selected with equal probability.

For the second part of this thesis, the assistant will also recommend the top-2 users as well as channels, which will be evaluated using the same live settings. Furthermore, the Recommendation model will be evaluated using a separate test set, retrieved using an export of the company's Slack posts the following month as the training and validation set was exported. This test will be compared to a naïve baseline of consistently recommending only two users and channels respectively; the two fixed recommendations that maximizes the score.

Since multiple answers might help a user find an answer, we will make the service recommend the top-2 recommendations found.

4

Experimental Setup

In this chapter we will discuss the experimental setup used to produce the results found during this project. We will describe an overview of the real-time system, as well as some details about the assistant service, implemented as a Slack bot.

4.1 System overview

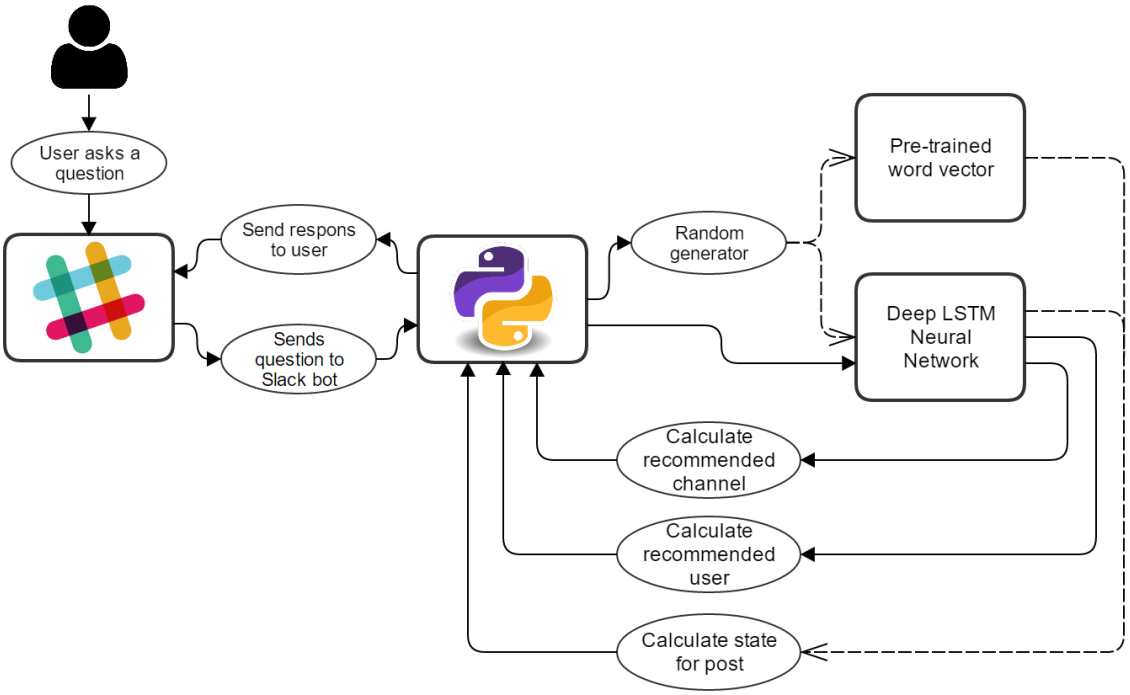

Figure 4.1: System overview of real-time system and flow

The central part of this thesis is the deep LSTM based neural network, which can be seen in the system overview in the top right corner in Figure 4.1. However, in the overall system that is being developed, it is just a tool to be used in order to provide the Slack bot with its main feature: recommendations. Otherwise central parts of the system are the Slack API and the Slack bot, which handles all the communication to and from all the available Slack clients (web, desktop, mobile).

The flow of the system can be described in the following 5 steps:

1. A user posts a question into a Slack client
2. The Slack API sends the user's question to the Slack bot
3. The Slack bot queues three jobs to the LSTM network:
   (a) User recommendations
   (b) Channel recommendations
   (c) Randomize, with equal probability, if the LSTM should be used to calculate the LSTM state of the post or if the pre-trained word vectors should be used.
4. The three jobs are independently returned to the Slack bot, which creates a Slack post as response for the three different recommendations.
   (a) For the user and channel recommendations, the bot will return with a simple message and attach the names of the users and channels.
   (b) For the post recommendations, the network or pre-trained word vectors returns the state for the post, which the bot then uses to compare the post with all other public posts on Slack.
5. All three recommendations are independently returned to the user, along with the encourage to react to all the recommendations independently.

4.2 Slack API and Slack bot

Slack provides multiple API interfaces in order to build Slack applications. For this thesis two of these were used; The Web API and the Real Time Messaging (RTM) API.

The Web API is used as a static API, where we have the ability to retrieve information about users, channels and also do a series of functions to the different concepts around Slack, such as invite, remove, create, rename a channel and delete messages in various channels (Both public, private and direct messages). This API is mostly used to get more information about a channel or user, in case new channels or users are created.

Listens to	**Used for**	**Slack RTM Event Type**
New channel creations	To get notified when a new channel is created on the company's Slack	channel_created
A user is typing a message	To get notified when a user is starting to write a new Slack post	user_typing
A user reacting to a Slack post	To get notified when a user has reacted to a Slack post	reaction_added
New messages posted to Slack	To get notified when a new Slack post has been posted	message

Table 4.1: Showing the different events the Slack bot listens to

More actively used is the RTM API. This is what the real time service uses to interact with Slack in a real time fashion. The Slack bot is connected to a real time session, which allows the bot to listen on a number of different message types, shown in Table 4.1. The most important event is the *message* event. This event is what

causes the Slack bot to be notified of new messages.

A Slack bot has somewhat restricted access to the Slack APIs. Two of the most troublesome restrictions are the inability for a Slack bot to join a channel by itself. It needs to be invited by a human. This in turn is necessary since a Slack bot cannot listen to messages sent, even in public channels, in which it is not a member. This is the purpose of the *channel_created* event. In order to let the bot be as productive as possible, and help people across different channels, it needs to be a member of as many channels as possible. In order to let the Slack bot be a member in channels that can be created in the future the *channel_created* event informs the bot about a newly created channel. Most importantly, the name of the new channel, as well as the Slack user ID of the creator of the channel is sent to the bot. The bot then stores this information and starts a direct message with the channel's creator, introducing itself and asking the creator for an invite to the channel. This lets the Slack bot to spread its information without any troublesome interaction.

Positive reaction	**Emoji**	**Negative reaction**	**Emoji**
smile	😄	disappointed	😞
wink	😉	neutral_face	😐
+1	👍	unamused	😒
simple_smile	🙂	worried	😟
upvote	⬆	confused	😕
smiley	😃	slightly_frowning_face	🙁
grinning	😀	white_frowning_face	☹
sunglasses	😎	-1	👎
clap	👏	downvote	⬇
ok_hand	👌		
amazed	😲		

Table 4.2: List of reactions the Slack bot will recognize

For the user satisfaction study, the bot can automatically identify if a reaction made by a user is positive or negative. The *reaction_added* event is used to let the Slack bot get direct feedback from the user about the recommended content. The bot will recognize a number of different reactions. All available reactions that automatically can be identified can be seen in Table 4.2.

4.2.1 Training data

The total number of Slack posts used for training was 184637. The company's Slack posts are often very short, with an average and median length of 17.42 and 11 tokens respectively.

Number of posts	Shortest	Longest	Average	Median	90th %ile	99th %ile
184637	1	1008	17,42	11	37	105

Table 4.3: Slack posts meta data

Table 4.3 shows some metadata about the Slack posts used for training.

For the second part of this thesis, the Slack posts was modified in order to give better results. Instead of working with 184637 posts, some posts was at random duplicated and some was altered. This was done in order to reduce the gap between channels with a lot of activity, compared to channels with less activity. This is described in more detail in Section 3.5.2

4.3 Training the First Model

During the training of the model used in part 1, early stopping (See Section 2.4.3) will be used to determine the number of epochs trained (See Section 2.3.1). For this, both training cost, as well as a validation cost will be gathered in order to evaluate the training. Also, for a more empirical study of improvements, the network will be sampled before and after each epoch when training using the Slack dataset. For the Wikipedia data set, the network will be sampled after a fixed number of iterations. Since the Wikipedia data is so large, especially compared to the Slack data, only one full epoch will be run. Instead of indicating the results as before and after an epoch, it will be indicated as before and after a file. In total the Wikipedia dataset will be split into 429 files.

4.4 Hyper parameters

There are multiple parameters used for the models, both the model used during part 1, and even more for the Recommendation model used in part 2. Here, a handful of parameters will be presented. As mentioned in Section 3.4.2, the models are inspired by the results obtained by Zaremba et al., 2014 [28].

The first model, as well as the first parts of the Recommendation model, uses 650 hidden units in the LSTM cells. The depth of the networks is 2. These values was used to match the results achieved by Zaremba et al. (See Section 3.4.2). For the pretraining phase, the output layer of the language softmax layer is 73949 outputs. This is the number of words in the vocabulary. For the Recommendation model, the outputs of the user and channel softmax layers are 660 and 321, respectively.

More parameters include the already mentioned cut-off parameter (See Section 2.5.3), which is different depending on which data set it is used on. For the Wikipedia data set, it was set to 50, while it was set to 2 for the Slack data set. These will be further described in the Result chapter.

The dropout used, which is one of the regularization techniques used, was set to 0.8, meaning it is a 80% chance that the values of a specific cell will keep its training. This value was decided empirically by training the first network, and checking how fast and low the validation cost reached.

5
Results

In this chapter the results from the two parts of this thesis will be presented. It will be presented in the order of which the parts was carried out. Before that, the results regarding the data sets used and the created vocabulary will be presented. This will be followed by the results after pre-processing.

We conclude that the models used during this thesis generalizes well, and is helpful for the users. The Recommendation model can give users quick recommendations in an instant messaging environment, recommendations that mostly is accepted by the users as good recommendations.

5.1 Vocabulary

As mentioned in Section 3.3, the final vocabulary was created using both the vocabulary created using that Wikipedia data set, together with the vocabulary created using the Slack data set. This is done since the training data consisted of both English Wikipedia articles, as well as the Slack data created by the company. Merging the two vocabularies resulted in a final vocabulary used of size 73949 tokens.

5.1.1 Wikipedia Vocabulary

When creating the Wikipedia counter, the final word count was 1001530 unique tokens. This is the counter created in step 4, described under Section 3.2.1. One of the least used word is *darmarän* and the most used word is *the*.

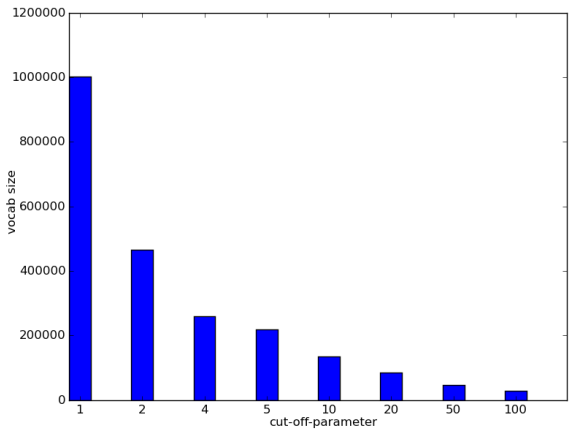

Figure 5.1: The size of the Wikipedia vocabulary decreases as the cut-off parameter increases

The cut-off parameter for the Wikipedia counter was selected to be 50. This was determined using empirical studies where different values of the cut-off parameter was used. Figure 5.1 shows the different values of the cut-off parameter that was tested and the resulting size of vocabulary.

wooda	secessionists'	chegorin	phintella	郑
darmar ā n	glorye	occipitofrontalis	みきこ	herasymyuk

Table 5.1: Sample of words that only occur once in all the Wikipedia articles

Two key aspects was considered when selecting the value of the cut-off-parameter. The least used words still included was examined and determined if they seemed relevant or not, as well as the final number of tokens in the set. The balance between having as few words as possible to speed up the training and the network's ability to generalize good, and including enough words for articles not to include too many UNK-tokens was considered. Table 5.1 shows some examples of words that only occur once in all the Wikipedia articles. As can be seen, the words that only occur once are either very rare words, words that are misspelled or words that are not English. Since these words only occur once in a dataset of this size, their relevance are very low. In total, there are 534824 tokens that only occur once in the Wikipedia data set. This is about 53,4% of all the tokens.

As seen in Figure 5.1, the size of the vocabulary quickly decreases when the cut-off parameter was increased. The biggest change is when changing the cut-off parameter from 1 and 2, where the size of the vocabulary changes from 1001530 words, down to 466706 words. Filtering out more than 50% of the words, when using a cut-off-parameter value of 2, still leaves a lot of strange words, such as *parcheggi*, *pujyapada*, *guirior*, *jähn* and *lucaswolde*. We finally decided to use a value of 50 for the cut-off-parameter. This gives us a vocabulary size of 46898 tokens and one of the least occurring tokens such as: *perfumes*, *refreshed*, *malfunctions* and *reformatory*, which could still be considered reasonable words to have in the vocabulary

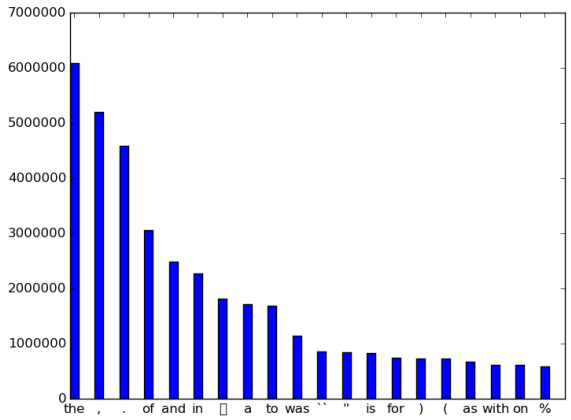

Figure 5.2: Word occurrences in the Wikipedia vocabulary

Table 5.2 shows the occurrence of the 20 most frequent tokens within the Wikipedia vocabulary. As can be seen along the horizontal axis, there is not only words, but also different signs such as commas, certain Asian characters and parentheses. The

distribution over the values on this table, as well as Table 5.3, is known as Zipf's Law, which is an empirical law that describes the probability of encountering the nth most common word in a large enough set of words [1]

5.1.2 Slack vocabulary

One of the biggest problems with the Slack data is that it doesn't solely consist of English words. The company, from where the Slack-data is gathered, is situated all over Europe, with their main office in Sweden, but also have offices in Germany, France, England and Norway, to name a few. Since the company has dedicated channels for each office, more often than not, the native tongue is used within that specific channel. This means that going through the Slack counter (See Section 3.2.2), words both in English, but also in Swedish, German, France etc. will occur.

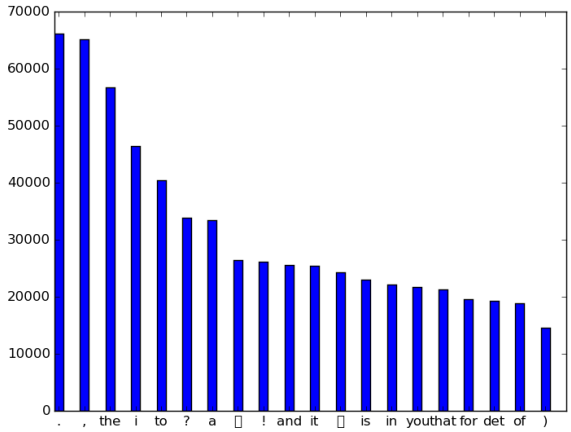

Figure 5.3: Word occurrences in the Slack vocabulary

Without filtering, there exists 95592 unique tokens in the Slack data. Here the most used token is "," (a comma) which occur 65103 times, and one of the least used tokens is *extraförställningar* which only occur one time. Figure 5.3 shows the number of occurrences for the 20 most used tokens in the Slack vocabulary.

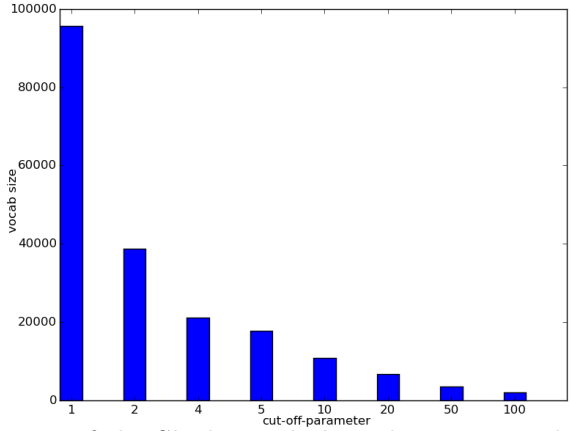

Figure 5.4: The size of the Slack vocabulary decreases as the cut-off parameter increases

[1] `http://mathworld.wolfram.com/ZipfsLaw.html`

The vocabulary created using the Slack-data is not as diverse as the vocabulary created using the Wikipedia data, so the same value for the cut-off parameter cannot be used. Words such as *singleplayer* and *headphone* is considered normal words for a company engaged in IT, but those words still only appear twice in the vocabulary created by the Slack data. The same principle used when creating the Wikipedia vocabulary to decide the value of the cut-off parameter was used with the Slack vocabulary. The cut-off parameter was decided to be set to 2. This resulted in a vocabulary of 38836 tokens. Figure 5.4 shows how the size of the vocabulary decreases as the cut-off parameter increases.

5.2 Training the Language Model

File	Before epoch	After epoch
1	the hennes datumet trick jazzy selger shawl onykter incised iterate roving	the different of the “ barletta „ giordano was above-ground
2	the guayaquil men by a film with the braves phenomenology of	the states using late only surface spöa to fulla by august
3	the depression each for pled men . reg cross-country gusts children	the war of properties , those brush in superpepp each way
205	the hague as pedestrian under the new name , present-day ye	the previous year professing humble barks , with a gifted municipality
206	the races that have favored peter carrington to concessions to the	the emperor csgo shop access method , for a performance account
207	the mantra — when the charmed rod noted the pressure in	the road , and named “ new york exchange ” .
427	the supercharger hole . gcc goodrich railroad the call-up to providence	the track and the latter both an austell grammar and strap
428	the source used by a contemporary magazine . the term “	the ’s common hall (created by pioneer elizabeth r. fenton
429	the afl fighter in a brock stadium out by the united	the joho stadium , where hill was located near the city

Table 5.2: Sampling before and after training runs using Wikipedia articles

Epoch	Before	After
1	the theory area of the woodstock had been adopted , a	the calendar is unittest , clever , major , but to
13	the inner valley neighborhood motorway . small pools will be a	the guys is people running on the hotel underground ? why
26	the markus spec apis are a bit late - i just	the process of the holocaust conviction is about industrialization . current
39	the most popular thing is the buzzword . i can not	the team requirements include almost root disturbances .,kochi

Table 5.3: Sampling before and after each epoch using Slack data

The two tables, Table 5.2 and Table 5.3, shows the sampled sentences created by the network during the training, both for the Wikipedia data set as well as the Slack data set. The network was configured to sample 10 tokens, using *the* as prime. Note that all the sentences have 11 tokens in them. That’s because the network does not count the prime. This is only to pre-feed the network into the desired state before sampling.

In total the network was trained for 39 epochs using the Slack dataset.

5.2.1 Cost

Here follows four plots of the cost during the training of the first model. The cost is the cross-entropy loss described in Section 2.3.1. The data for the plots are fetched

using Tensorflow's Saver class[2] and then plotted using Tensorboard, a Tensorflow visualization utility program [3].

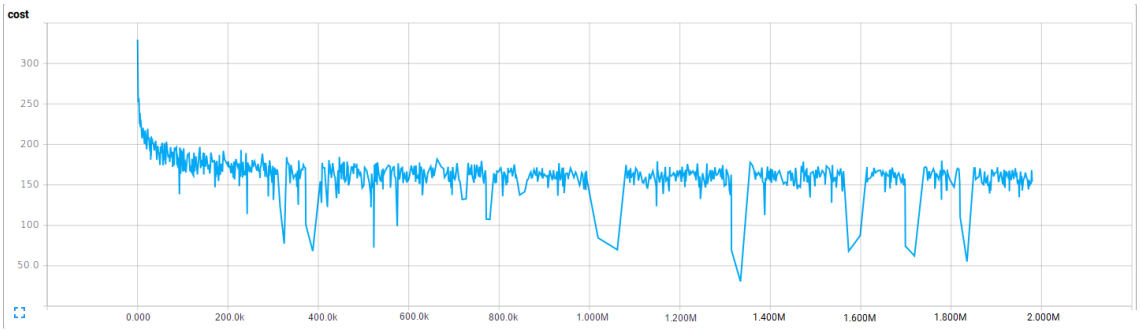

Figure 5.5: Training cost, for one epoch using Wikipedia data for part 1 of this thesis

Figure 5.5 shows the training cost when using the Wikipedia data over the whole epoch

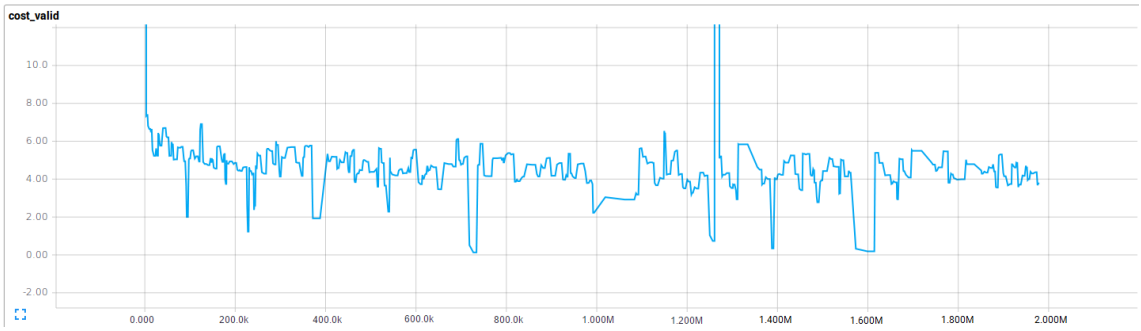

Figure 5.6: Validation cost, for one epoch using Wikipedia data for step 1

Figure 5.6 show the validation cost when using the Wikipedia data. Interesting to note in this plot, when comparing to Figure 5.5 is that the network does not seem to overfit, since the validation cost is slowly decreasing and in particular not increasing while the training cost decreases. The final validation cost for the Wikipedia data set run was around 3.7

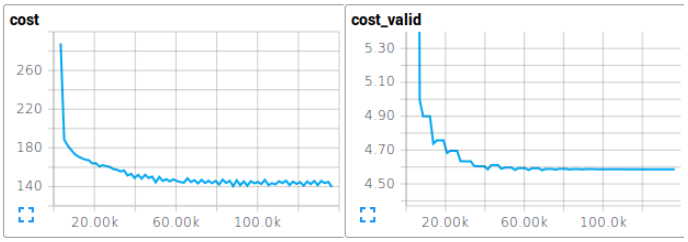

Figure 5.7: Training cost and validation cost over steps using Slack data for step 1

After the Wikipedia data set has been used for training the network, the Slack data was trained for 39 epochs. Figure 5.7 shows the training cost over all epochs (left

[2] `https://www.tensorflow.org/versions/r0.7/how_tos/variables/index.html`

[3] `https://www.tensorflow.org/versions/r0.7/how_tos/summaries_and_tensorboard/index.html`

plot) as well as the validation cost (right plot) over the same period. The network did validation after every even epoch. Also here, as with the plots for the Wikipedia data, the validation cost decreases together with the training cost. This is a good indication that overfitting did not occur. Note that the different scales on the two plots are due to the different batch size used for training and validation.

5.3 Finding Slack posts

The first part of this thesis was about finding similar Slack posts, given another Slack post. This was done in two steps: using pre-trained word vectors and using the last state of an LSTM based model, pre-trained as a language model.

Cosine	Pre-trained word vectors
0.854	Having a edge on differen javascript frameworks would be very cool. one per technology :simple_smile:
0.848	So I have a lot of javascript that will be used across about 40 sites. The javascript contains a lot of mandatory, minimum functionality for tracking user behaviour. But to fit a variety of additional needs that each individual site has, I want to make the tracking part of the javascript extendible/pluginable. The goal is that the sites can both add data to automated tracking provided by the javascript, and track events that are not automatically tracked by my javascript. So I was wondering if anyone has any good articles outlining a simple and/or good plugin architecture for javascript. Or even better, if anyone in, has done something like this before?
0.842	Hey guys! Me myself and,are having a discussion regarding using Typescript with Angular.js or not. Can you share knowledge like adv. and disadv. of using plain Javascript or Typescript! Our client uses Typescript so we want to use the same, to maximize compatibility with their way of doing web development. A good update on the two would help us in making sure we can develop fast and without delay!
0.842	<user>: Media queries would not be great as well as you said. Are you using any Javascript libraries in your project that could help? Angular maybe?

Table 5.4: Top 4 responses from the baseline method (See Section 3.4) when asking the question "Do we have any experience with using angular and javascript two way databinding?"

Cosine	Recurrent Forum Assistant
0.927	can someone recommend testing frameworks for Python?
0.921	Does anyone have experience in using Zend Server (for debugging) with Eclipse?
0.918	are you using any framework? such as phpspec?
0.918	To add on the framework question, are there some libs for workflow systems maybe even with BPEL support?

Table 5.5: Top 4 responses from the recurrent forum assistant (See Section 3.4) when asking the question "Do we have any experience with using angular and javascript two way databinding?"

The results in Table 5.4 and Table 5.5 are the 4 most similar posts found together with their cosine similarity score (See Section 2.6).

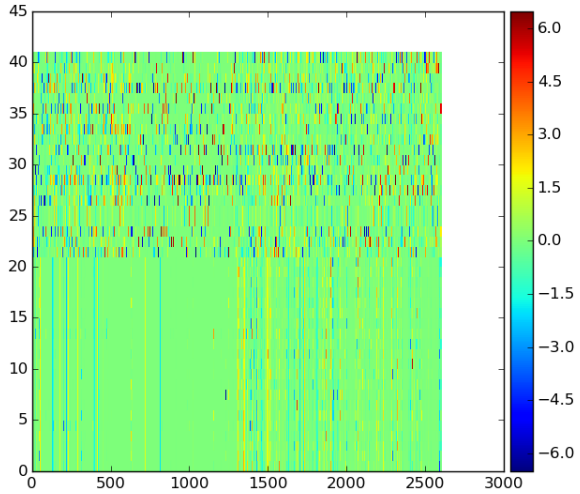

Figure 5.8: Heatmap plot for the 20 most similar, 20 least similar and the original text

Figure 5.8 shows a heatmap plot over 41 posts' LSTM state representation. These are the last states of the LSTM network. The posts are laid along the Y-axis, where the first row is the input post. After the input post, the 20 most similar posts are plotted, and lastly the 20 least similar posts are plotted. The X-axis shows the size of the LSTM's state (2600 dimensions).

5.3.1 Clustering Slack posts

Also done using both the pre-trained word vectors and the states from the LSTM network, was to plot the vectors using t-Distributed Stochastic Neighbor Embedding (t-SNE), a prizewinning technique well suited for visualizing high-dimensional datasets[4]

Unfortunately, due to the number of posts present in each channel, it was not feasible to plot all available Slack posts. Insead, plots was done for each individual channel. As a result of the way the t-SNE works, the resulting plots cannot be compared between each other. They are fully relative.

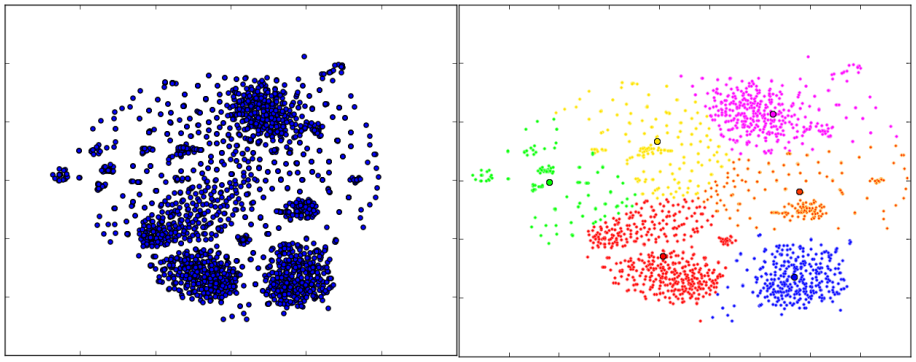

Figure 5.9: t-SNE plot over all posts within a mobile development channel using LSTM states to the left, and the same posts to the right, but clustered using k-mean

[4]http://blog.kaggle.com/2012/11/02/t-distributed-stochastic-neighbor-embedding-wins-merck-viz-challenge/

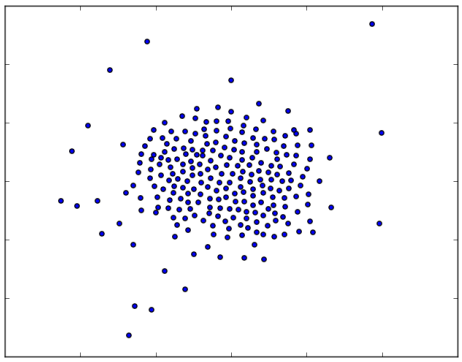

Figure 5.10: t-SNE plot over all posts within a mobile development channel using pre-trained word vectors

Both Figure 5.9 and Figure 5.10 are plots of all Slack posts within the same channel, a mobile development channel. Figure 5.9 has as input the 2600 dimensional vectors retrieved from the LSTM's final state, whereas Figure 5.10 uses the 300 dimensional vectors obtained using the pre-trained word vectors. The right plot in Figure 5.9 has the scatters colored depending on which of the 6 clusters it belongs to, according to a k-mean clustering algorithm [5].

Color	Closest to cluster center	2nd closest to cluster center
Green	Maybe the Asana-math font will work: <link1><link2>	xposed for lollipop is finally out! <link>
Yellow	Google Play beta	Im not using the beta
Purple	ok..	well summarized <user>
Orange	Hi all!	Hello HK!
Blue	does anyone know how you figure out the distanse between a android phone and a beacon?	can that push the app to my phone?
Red	Ok, I was hoping to do that java->;c++ call via some function pointer or whatever, so that I would not have to create a new function in jni which would be explicitly known by the java code.	Yeah, im using retrofit as well. Planning on how to use the Callback paramter. Directly to the activity or through som bus. So now its either the Eventbus or RxAndroid for that purpose

Table 5.6: The closest and 2nd closest posts to the 6 different cluster centers shown in Figure 5.9

Table 5.6 shows the different types of posts that occur in the different clusters that is shown in the left plot of Figure 5.9. The posts shown in the table are the two posts that are closest to the cluster center.

5.4 Training the Recommendation Model

The second network used for this thesis is a continuation of the first one (See Section 3.5.2). The network was pre-trained as a language model, just as done during part one of this thesis. For this, the exact same parts of the network was used, as well as the pre-training. The training then continued with the specific users and channel layers, in order to let the network learn which user and channel to recommend. This training was done simultaneously.

[5] `http://scikit-learn.org/stable/auto_examples/cluster/plot_mini_batch_kmeans.html#example-cluster-plot-mini-batch-kmeans-py`

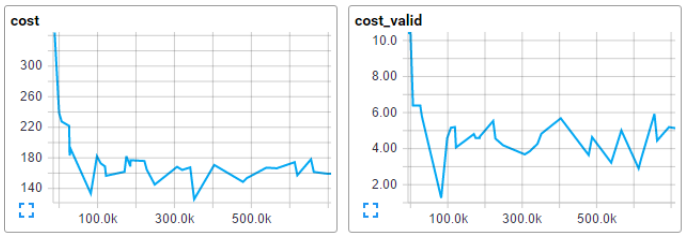

Figure 5.11: Training and validation cost for one epoch using the Wikipedia data over all steps trained for step 2 of this thesis

Figure 5.11 shows the training and validation cost when training the second LSTM network, during the pre-training process, when the network was trained as a language model. This is the first part of the pre-training, when using the Wikipedia data set.

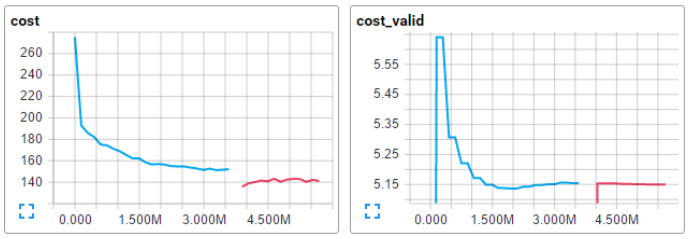

Figure 5.12: Training and validation cost for 39 epochs using the Slack data set over all steps done, for step 2

The second part of the pre-training for step 2 was to use the Slack data set. The training and validation costs can be seen in Figure 5.12. The different colors in the plots are due to a restart of the training. The training was abruptly interrupted in the middle of an epoch due to technical problems, and the training could be continued on the next coming epoch.

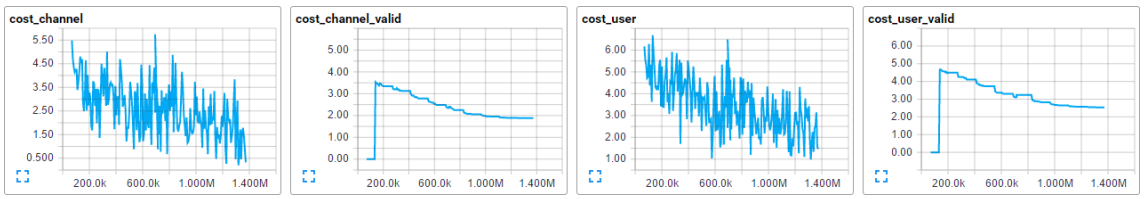

Figure 5.13: Training and validation cost when training for the 19 epochs trained for user and channel recommendations

Figure 5.13 shows the training and validation costs when training the Recommendation model, both for the user head as well as the channel head. The training was propagated all the way through the network, from the final Softmax layers, though the LSTM part down to the input layers (See Figure 2.4).

5.5 Recommend users and channels

Here the results from the live test of the Slack bot will be presented. The bot was set live in a special channel on the company's Slack called *quickhelp_by_bot*. Here

users could join and ask questions to the bot, which then would use the network to provide them with recommended users and channels.

	Positive	**Negative**	**No reaction**
Users	70.4%	6.1%	23.5%
Channels	80.9%	4.8%	14.3%
Posts LSTM	42.1%	47.4%	10.%
Posts W2V	35.7%	57.1%	7.1%

Table 5.7: Showing the results from the live user study. Percentage is based on the total number of reactions to the agent's actions (and an action from the agent that resulted in no reaction from users is counted as "no reaction")

Before the user satisfaction study was performed, many small scale tests was done in order to ensure stability before putting to much load on the service. Table 5.7 shows the results from the user and channel recommendations made by the bot and the user's reactions. For users and channels recommendations most reactions are positive, suggesting that our assistant is useful to the forum users. In total, 123 reactions were collected in the user study.

	User	**Channel**
Recurrent assistant	14.39%	22.01%
Naïve baseline	2.46%	5.54%

Table 5.8: Accuracy of the recommendations from the agent regarding Slack users and channels, respectively, on the separate test set

Table 5.8 shows the results from the evaluation of the Recommendation model when testing the separate test set (See Section 3.6). The proposed assistant beats the naïve baseline by a large margin.

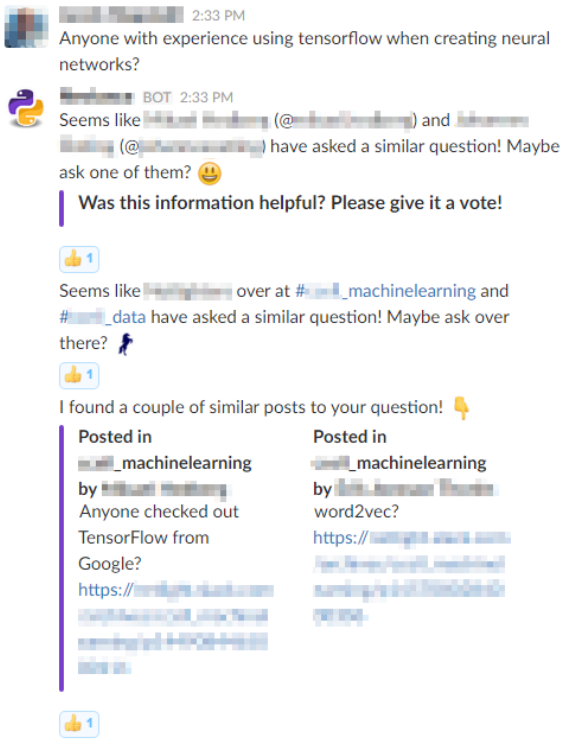

Figure 5.14: Screenshot of the Slack user interface when asking a question. Names and usernames have been anonymized

Figure 5.14 shows how the Slack bot gives its three recommendation types when a question is asked.

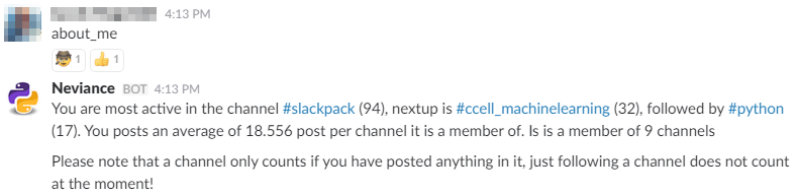

Figure 5.15: Screenshot of the Slack user interface when asking about what the user talks about

The Slack bot also provides the user with information about which channels the user is most active. This can be seen in Figure 5.15. As can be seen, the bot also informs the user that a channel only counts if the user has posted anything in the channel. A user can follow multiple channels, but if she/he never posts anything, those channels will not count.

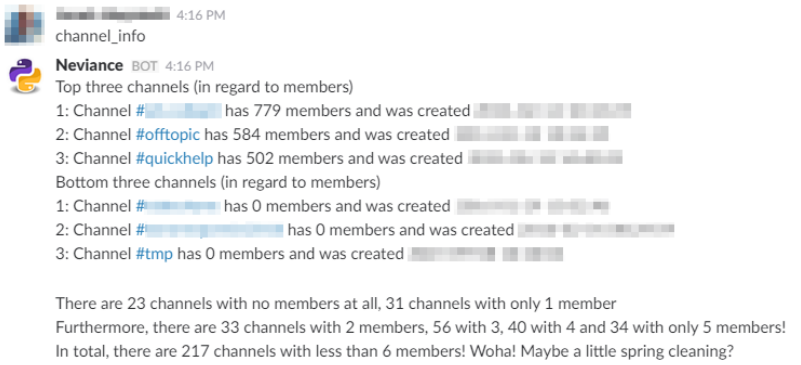

Figure 5.16: Screenshot of the Slack user interface when asking about the most and least popular channels

It is also possible to ask the Slack bot about the most popular channels, in regard to membership-count. An example of this can be seen in Figure 5.16. The bot also informs the user about the number of channels with a low membership-count. This can be of use if removing old or obsolete channels is of interest. If the bot finds that it exists channels with either no members or a channel with a low number of channels (≤ 5 members), it will inform the user that *some spring cleaning* can be done.

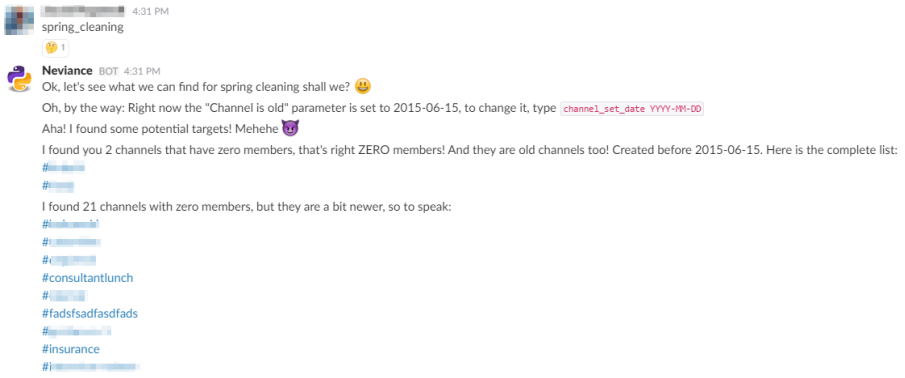

Figure 5.17: Screenshot of the Slack user interface when asking the bot to list channels which can potentially be removed

Figure 5.17 shows how the Slack bot can help to inform users (preferably admins) which channels might be obsolete and can be archived.

6
Discussion

This chapter will do a more in depth discussion about the results presented in the Result chapter and how the networks have performed. We will also connect back to the initial demo made in very beginning, to compare how a more advanced LSTM network performs against a more simple pre-trained word vector model.

The initial result using the pre-trained word vectors revealed that the hypothesis of how to find similar posts, namely to use cosine similarity, worked and that contextual similar posts actually was returned by the demo. When the first LSTM network was built and trained, we changed all the posts vectors into the final state of the LSTM network, and the results became a bit less specific compared to when we used the pre-trained word vectors. This seems like a reasonable result from a network that was trained as a language model. E.g: a language model will compute a similar distribution over the next word after observing the word "Python", as compared to observing the word "Java".

The results for the user and channel recommendations was very positive, both from the live user study, as well as running the test set through the Recommendation model.

6.1 Training Data

When the final vocabulary was created, a random Slack post was selected and all tokens contained within the Slack post not in the vocabulary was replaced with a "<unk>" string. When the post was later examined, it showed that we don't seem to need a lot of tokens in order to form the most used sentences. This reinforced the belief that the size of the vocabulary was sufficient in order to capture the most important information about the Slack posts.

When tested, a smaller vocabulary allowed the network converge faster, and had a better ability to generalize. This gives us confidence in the choice of value for the cut-off parameter. A larger vocabulary would also have become troublesome to fit on the GPU memory.

6.2 Training the LSTM

The training of the LSTM networks used during this thesis will under this section be divided into the two parts that was conducted. The two networks created does not differ too much, but the way they are trained and the results they give are.

6.2.1 Part 1

The first LSTM network trained had the purpose of giving us its final, internal state, to be used as a mathematical representation of a Slack post. In order to get feedback on how the network performed during the training both the training cost, as well as the validation cost was plotted and examined. The cost plots during the training using the Wikipedia data set (See Figure 5.5) showed that the cost seemed to have a hard time to decrease. Some articles made the cost decrease lower than others. This was unfortunately not a lasting trend, and the next article used made the cost return.

Due to the time limitations of this thesis, and the amount of data used for pre-training, namely the entire English Wikipedia, only one epoch was trained. Thanks to the positive results given by the network, at the end of the training, we believe that one epoch was enough to let the network learn about generic English language. The network had grasped some general ideas about the language, and instead we could let the network train on the Slack data set. Here the costs continued to decrease, and after the 39 epochs of training it started to give us very nice results (See Figure 5.7.

Another point that made us believe that the network performed rather good, was the sampled sentences. Both the sentences sampled during training (See Table 5.5), but also sentences sampled afterwards using the Slack bot. The sentences made sense and the network correctly inserted both starting and ending parentheses, but also quotation marks, exclamation marks etc.

6.2.2 Part 2

The second LSTM network had the purpose of recommending users and channels, given a Slack post, as well as provide its internal state to find similar posts. The training, as explained in the Method section, was fairly straightforward. The initial thoughts was to not back-propagate throughout the entire network, but to keep the LSTM part of the network untouched, and only train the Softmax layers (See Figure 3.5. This was done so that the language part of the model would not be destroyed after the pre-training, so that the same model could be used for all three recommendations: similar posts using the internal state of the LSTMs, as well as user and channel recommendation using the Softmax-layers.

Unfortunately, this proved to be unsuccessful, and the during the first live test of the bot, employees from the company did not feel that the users and channels returned was a good recommendation, given the question. Instead, a decision was made to

continue with the backpropagation all the way through the network. This gave us very good results for the users and channel recommendations. This can be seen in Table 5.7. Also noticed during the live tests of the bot, the recommended posts seemed to be better for the Recommendation model, than the first model. The network seemed to have learned a better representation of the input. This made it possible to use the same model for all three recommendations

6.3 Finding Slack posts

When starting this thesis, the expectations made by us was set very high. We expected a well working deep neural network, and that the posts found would have high relevance for the questioner. After seeing the initial results using the pre-trained word vectors, these assumptions grew stronger. When the first results was produced by the network, our initial response was that the network performed really bad in a sense, and that the idea of switching the word vectors to the internal state of the LSTM might not have worked as intended. This started an investigation where we tried to determine what was wrong, and where the problem lay.

The first thing that was considered was that the network wasn't trained well enough. The sampled sentences (See Table 5.2 and Table 5.3) the network generated during the training indicated that it was working as intended; the sentences became better and better in the sense that the sequence of words sampled made more grammatical sense. Furthermore, the heatmap (See Figure 5.8) together with the corresponding texts of the posts indicates that the posts, that the network said was similar, actually was similar in a context based view. The examples in Table 5.5 shows that all posts found have something to do with programming, it is often someone asking for experience/recommendations just as the question. However no keywords from the original question are present. This is what triggered the initial reactions that the network performed bad. But after careful inspection, the network performed exactly as intended. It did manage to retrieve posts that has similar context. It could be an easy add-on to also post-process the results by filtering out posts that do not have some specific keywords in them, given a list of keywords from the question. This is unfortunately outside of this thesis' scope and will not be examined. This is also why the results using the pre-trained word vectors in a first glance seems to give better results. Having the actual word in the post as with the question will shift the post's vector into that direction, whereas the network looks at a bigger picture, trying to generalize about the context of the question and the found posts.

From the test results, the results from part 1 of this thesis can be considered good and the network finds context similar posts. Given the example sentence used (See Table 5.5):

- All posts returned are questions
- All posts have something to do with a recommendation, or an opinion about an experience
- They all have something to do with programming in one way or another
- This is what could be expected by a network that is trained in this way

The network is firstly trained as a normal language model for the English language, without any bias towards specific terms encountered in the IT world. The network is then trained using the company's Slack data in order to bias it a bit towards the way the company's employees expresses themselves. Despite the Slack data containing multiple languages, the network is able to perform when the question is asked in English.

6.3.1 t-SNE

t-SNE plots was made, in order to see if the plot's vectors retrieved using the LSTM network was able to distinguish the different topics talked about within a channel. As can be seen in Figure 5.10, when plotting all posts within a mobile development channel using the pre-trained word vectors, all posts seem to be spread out evenly, making it very hard to see any specific topics such as GUI, threading, database management etc. When instead plotting using the final state from the LSTM network, 4-5 big clusters are visible, together with some "noise" posts as well, and some smaller clusters. Now different kinds of discussions and posts are distinguishable.

Table 5.6 shows that when using the LSTM, we are able to distinguish between different types of text such as questions, links to outside resources (other web pages etc), simple greetings and other discussions about code. Further analyses of clustering posts has not been done, since that lies outside of the scope of this thesis. Instead more focus on user's experience when using the bot has been analyzed.

6.4 Recommend users and channels

This part of this thesis was evaluated using two techniques: A separate test data set, to see if the Recommendation model was able to give us the right author of the post, and where the post was made, as well as the user satisfaction study, described in Section 3.6. Something that was noticed during the first part of this thesis is that users have a tendency to talk about similar things across multiple channels. We believe this is due to several reasons:

1. The names of several channels are very general, and invites for a very broad discussion
2. Channels which has a more clear purpose still have a tendency to get out of scope: If a question or discussion that people find interesting is arising, chances that they will move the discussion to a more well suited channel is low.

The company has over 790 users, as of writing this report. This makes it very hard to evaluate if the network actually recommends people with knowledge about the question's topic. Also, since the final layer of the user and channel recommendation LSTM network is a Softmax, all the user's sureness will together sum to 1, making the network appear to be very unsure about a specific user it recommends when the topic is very broad. This, in contrast to the sureness of finding similar posts, where

the question is compared against all other posts individually.

All this means that several channels and several users might be the right one to recommend. Also channels that are very general might be recommended, when some users know that a specific channel for the question's topic exists, making them think that the recommendation isn't that good.

6.5 Slackbot

Since the main focus of this thesis was to explore the possibility to use deep nerual networks in an instant messaging environment, and not on a Slack bot, this section will not be so exhaustive. The bot is merely a smart way to let users interact with the neural network within the Slack client. An alternative could have been some sort of web-app, but that would have meant that users needed to go to a dedicated web site and ask their questions there. Instead they can use Slack as they already are used to, and they will get the power of a deep LSTM network.

Noted during the live tests of the bot, users formulated their questions differently when they knew that the bot was going to give them recommendations, compared to when they asked in a channel where other users was expected to give an answer. The questions was much shorted, more straightforward and sometimes they only wrote keywords and simply adding a question mark at the end. The Slack bot - human interaction perspective is a very interesting field, and should be further investigated. The way the networks are trained, makes the service perform worse when given keyword-questions and questions not formulated as the employees usually communicate within the company's Slack.

7
Conclusions

In this chapter some final words will be said about the overall result found by this thesis, as well as some thoughts of things to improve, and further research that can be made.

The LSTM networks modelled and trained during this thesis reflects state-of-the-art LSTM networks with very promising results. The results shows that it is possible to achieve good results in an instant messaging environment using deep learning. In order to make this more product-ready, some minor tweaks would need to be implemented.

7.1 Vocabulary

At the start of the thesis, a relatively big vocabulary was used. This later showed not to work, since the network had a hard time generalizing about the data, and to find patterns. Instead a much smaller vocabulary was used in the end, which turned out to work very well. Not only was this shown in the results the network produced (finding similar posts, sampling text etc), but it also made the training go much faster. This in turn made it possible to train the network for more epochs, resulting in a lower validation cost.

7.2 Training the LSTMs

Training deep LSTM networks has become fairly straightforward, much thanks to the improvements of the libraries found for training artificial neural networks, together with the ability to accelerate the training using much faster GPUs, compared to using only CPUs. Speed-ups between 20 to 30 times can be expected.

The cost plots when using the Wikipedia data set shows that both the training and the validation cost stagnates around 150-175 and 4-6 respectively. However, this could simply be because of the fact that the network was only trained for one epoch. Some of the articles seemed to be able to reach a much lower validation cost than others. It would be interesting to analyze what the affected articles cover to make the network perform better.

Nevertheless, the language part of the network seemed to perform well. The network is able to capture the general idea of the language: After seeing the sampled sen-

tences which very often not only is correct English, but also has its verbs conjugated correctly. Also, after seeing the results when finding similar posts (See Table 5.5), the posts retrieved is about similar topics, as discussed in Section 6.3.

7.3 Finding Slack posts

We have been able to utilize a deep LSTM, pre-trained as a language model, to find semantically similar posts. The results shown in Table 5.5 shows that the network is able to find what can be considered similar posts. At first glance, it could be argued that the network performs worse than with the pre-trained word vectors in the specific example shown in Table Table 5.4 and Table Table 5.5. We would rather argue that in this specific example the pre-trained word vectors has the advantage since a post that includes as much of the exact same words in a post will give it a high similarity. However, this is no better than using the already very good Slack search functionality, where a user can simply extract keywords from its question and search. With the final state from the LSTM network we give the user the ability to formulate the problem with its own words, and the network will still be able to find similar posts.

Worth noting is that despite the examples shown in the Result section, the network will not just return similar posts that has something to do with programming in one way or another. This is only due to the fact that the question asked had to do with programming. If a question regarding the best sushi lunch place in Stockholm is asked, posts regarding lunch, eating and fish will be returned.

7.4 Recommend users and channels

The results made by the Recommendation model is very satisfying, and we feel that our proposed assistant is useful for users and gives them good recommendations for both channels and users. The general impression is that users are more happy about the channel recommendations, in comparison to the user recommendations. This is also confirmed by the end-to-end evaluation made, using the unseen test-set (See Table 5.8). Our model learns to produce recommendations for both users and channels, and generalizes well to unseen data. The results from the user study clearly shows that the users find the suggestions from the assistant to be positive and useful (See Table 5.7).

7.5 Final thoughts

In this thesis, we have proposed a virtual assistant for discussion forum users, built using deep recurrent neural networks with LSTM cells. Our solution relies heavily on learning useful representations for the data in discussion forums.

We found that using the representations from a deep recurrent neural network can be useful for the retrieval of relevant posts. However, in this particular task we found that using a representation based on summing pre-trained word vectors works comparably well and gives more specific responses.

7.6 Improvements and future research

There are interesting aspects and ideas that wasn't part of this thesis. These are left for future research. Most of them are left due to time limitation. Here follows some selected improvements and future research that can be made.

7.6.1 Human interaction

Noticed during the live test of the assistant, was the problem that users knew that they were asking a question to a bot, making them formulate their questions in a shorted, more concise way, compared to when they asked questions to other users. Due to the way the networks used in this thesis was trained, this resulted in degraded recommendations. This poses the question, how can a bot present itself so that it feels more natural to talk to it?

7.6.2 Post-processing

The model used in part 1 of this thesis produced final states for all of the company's Slack posts in a way that made it possible to find semantically similar posts using cosine similarity. This worked exactly as intended and we found the results to be as expected. However, for a Slack bot in production, users will most likely not be happy to get a recommendation to another Slack posts which, for example, also asks a question about databases, unless it is about the same kind of database. For this to be possible, some kind of post-processing needs to be done in order to filter out posts that does not mention the right kind of database. This will require more manual work, in order to generate a list of keywords for every kind of text input, but the results will hopefully be less generic and be more like a question that a person would say is similar to the original question.

For this thesis, the Slack bot returns the posts found, together with a direct link to where the post was made (See Figure 5.14). The intended usage was to let the user go to the post they found as a match, and scroll down in the Slack history, in hope of finding an answer to their original question. This is something that the bot could be doing instead. For this to work, some sort of matching system needs to be implemented, in order to identify a possible later answer to a question. Here an answer can also be posted long after the question was posted, meaning there is a possibility that other posts regarding totally new questions have already been asked. This particular company often solves it by tagging the questioner together with the answer.

7.6.3 Change of targets

An interesting idea is to change the way the models are used to find similar posts (both the model used in part 1 of this thesis, but also how the second model was used). Instead of using the internal state of the LSTMs, the target can be an answer to a question asked in a forum. This might yield better results as a user expects when it comes to finding similar posts.

7.6.4 Identify discussions

During this thesis, it has been noticed that the company's Slack usage in the public channels comes in bursts. When an interesting topic arises, many users join the conversation. If these "mini discussions" could be identified, the network could get more data for training the user and channel recommendation layers. Not only does a user have to write in a certain way, but it can also be a target for the kinds of discussions it engages itself in.

7.6.5 Identify questions

For this thesis, a Slack bot is reacting to a question simply by using regex, where it searches for a question mark in all the posts posted in the channels on Slack where the bot is a member of. What if it could be possible to isolate what, in the final LSTM state representation, defines a question? If so, then subtract this using simple vector algebra is order to convert a text between a question and a statement?

This could also be a better way of letting the bot know when to start looking for recommendations, since it might make the bot not react on every single question mark, but only when an actual question is asked. This would allow users to still use the actual questionmark character, without the bot needing to give any results

7.6.6 More team specific vocabulary

When the vocabulary used in this thesis was created, it was composed from two different data sets. A Wikipedia data set, with the intention of getting words that builds up the core language of the English language. Then a vocabulary created from the company's Slack data was added in order to also get hold of some specific words used by the company's employees.

This could be done even more detailed, where specific words used by a specific company and industry can be added, even if it might not already be occurring enough in order to clear the cut-off parameter (This parameter is further explained in Section 2.5.3). This would make the network understand more about what a specific usergroup talks about and even if an unusual word, such as the name of a specific IT framework appears, it would not be changed into a UNK-word.

7.6.7 Pre-training the language model

Wikipedia is extremely nonuniform, so to shuffle the data before using it to train the language model could be a very good idea in order to achieve better results [23]. Also it would be very interesting to see how the networks perform if longer Wikipedia training was done. This was not done during this thesis due to time limitations.

7.6.8 Hyper parameter testing

There are a lot of parameters used by the networks used in this thesis. Only to configure the network there are about 19 parameters, setting the size of the hidden layers, setting the learning rate decay etc. In addition, parameters such as the cut-off parameter when creating the vocabulary exists (See Sectino 4.4).

All these can be modified and might either give a better result, or make the network converge faster. For further studies, analyzing which size of the hidden size is best, what learning rate makes the cost decay fastest etc could be conducted.

7.6.9 Utilize Slack specific syntax

Slack uses some special syntax in their messages. This include tagging of other channels and also users. We have also noticed that, at least for this company, the vast use of emoticons in messages and also "reactions". A Slack reaction is where a user has the ability to tag a post with an emoticon. All these special syntaxes could be possible to utilize, either by some post-processing, or better still: by letting the network learn when to use emojis, and when to tag a user or channel. As an example of this, using the fact that users can be tagged in a message could be in order to learn who to recommend given a post. Also messages with a lot of reactions might be interesting to recommend, since it indicates that many users think that post is good.

7.6.10 Other kinds of recommendations

In order to make this a better product, improvements not directly connected to the neural network can be implemented and tested. This include, for example, a more advanced recommendation system.

During this thesis, the only kind of recommendations used was the ones directly produced by the LSTM network. In order for a user to get more, and possibly better recommendations, is to implement a more advanced recommendation system. This can include:

- Recommend users given user(s)
  - If user A and user B often talk to each other, or are members of the same kinds of channels, but user B is also talks a lot to user C, then user C might be a candidate for recommendation for user A
- Recommend channels given user(s)

  - If user A and user B often talk to each other, or are members of the same kinds of channels, but user B is also a member of a channel that user A is not, then this channel might be a candidate for recommendation for user A
- Recommend users given channels
  - This is strongly related to the currently implemented recommendation system, but instead of just recommending a user given the user's posts, we also give the user a strong connection to the channels where the user often make posts

7.6.11 Reinforcement learning

The Slack bot can handle Slack reactions to its recommendations. This is how users can give feedback to the bot, by giving the bot reply a positive or negative reaction. The reactions the bot is able to understand can be seen in Table 4.2 under the Experimental Setup section. A very interesting continuation of the feedback system would be to let users edit the bot's answer, to give suggestions on which channel or user that they believe would be a good match.

Bibliography

[1] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[2] Y. Bengio, H. Schwenk, J. Senécal, F. Morin, and J. Gauvain. Neural probabilistic language models. In *Innovations in Machine Learning*, volume 194 of *Studies in Fuzziness and Soft Computing*, pages 137–186. Springer, 2006.

[3] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[4] H.D Block. The perceptron: A model for brain functioning. i. *Reviews of Modern Physics*, 34(1):123–135, 1962.

[5] D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333 – 338, 2012.

[6] Y. Freund and R.E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

[7] A. Gandhe, F. Metze, A. Waibel, and I. R. Lane. Optimization of neural network language models for keyword search. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2014, Florence, Italy, May 4-9, 2014*, pages 4888–4892, 2014.

[8] F.A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451–2471, 2000.

[9] Felix A. Gers and Jürgen Schmidhuber. LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Trans. Neural Networks*, 12(6):1333–1340, 2001.

[10] A. Graves. Offline arabic handwriting recognition with multidimensional recurrent neural networks. *Guide to OCR for Arabic Scripts*, pages 297–313, 2012.

[11] A. Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[12] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[13] A. Kumar, O. Irsoy, J. Su, J. Bradbury, R. English, B. Pierce, P. Ondruska, I. Gulrajani, and R. Socher. Ask me anything: Dynamic memory networks for natural language processing. *CoRR*, abs/1506.07285, 2015.

[14] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[15] T. Mikolov and G. Zweig. Context dependent recurrent neural network language model. In *2012 IEEE Spoken Language Technology Workshop (SLT), Miami, FL, USA, December 2-5, 2012*, pages 234–239, 2012.

[16] M. Minsky and S. Papert. *Perceptrons: an introduction to computational geometry*. MIT Press, expand edition, 1988.

[17] K. Moritz Hermann, T. Kociský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1693–1701, 2015.

[18] D. Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann, 1999.

[19] T. Rocktäschel, E. Grefenstette, K. Hermann, T. Kočiskỳ, and P. Blunsom. Reasoning about entailment with neural attention. *CoRR*, abs/1509.06664, 2015.

[20] F. Rosenblatt. *Principles of neurodynamics; perceptrons and the theory of brain mechanisms*. Spartan Books, 1962.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[22] N. Srivastava. Improving neural networks with dropout, 2013. PhD thesis, University of Toronto.

[23] I. Sutskever, J. Martens, and G. E. Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 1017–1024, 2011.

[24] I. Sutskever, O. Vinyals, and Q.V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[25] D. Tang, B. Qin, and T. Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1422–1432, 2015.

[26] C. Wang, S. S. Venkatesh, and J. S. Judd. Optimal stopping and effective machine complexity in learning. In *Advances in Neural Information Processing Systems 6, [7th NIPS Conference, Denver, Colorado, USA, 1993]*, pages 303–310, 1993.

[27] C. Xiaodong, G. Vaibhava, and B. Kingsbury. Data augmentation for deep neural network acoustic modeling. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 23(9):1469–1477, 2015.

[28] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014.