CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Dialogue modeling using recurrent neural networks

Master's thesis in Computer Science

VIKTOR ANDERLING
JONATHAN ORRÖ

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

Master's thesis 2016

Dialogue modeling using recurrent neural networks

VIKTOR ANDERLING
JONATHAN ORRÖ

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Division of Computing Science
Machine Learning and Computational Biology
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2016

Dialogue modeling using recurrent neural networks
VIKTOR ANDERLING
JONATHAN ORRÖ

© VIKTOR ANDERLING, JONATHAN ORRÖ, 2016.

Supervisor: Mikael Kågebäck, 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
Division of Computing Science
Machine Learning and Computational Biology
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2016

Dialogue modeling using recurrent neural networks
VIKTOR ANDERLING
JONATHAN ORRÖ
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Abstract

As the importance of computers in everyday life increases, so does the demand for better human-computer interfaces. Natural language, being our most natural form of communication, combined with man's innate tendency of anthropomorphism, motivates the idea of a talking machine. Existing dialogue systems have the problem of being unable to answer out-of-domain questions as well as being tedious to design. While these systems are developed with hand-crafted rules, the goal of this thesis is to investigate if a dialogue system could be automatically trained to speak instead.

Our aim is to test whether a model trained on a dialogue corpus can compare to existing dialogue systems. We trained a recurrent neural network using the *sequence-to-sequence* method, preserving the state of the model during the course of the conversation. The resulting network is end-to-end trainable. User testing was used to evaluate the model and compare it to the other dialogue systems.

The final model can answer appropriately to common phrases such as greetings and valedictions. It also generates replies in correct English. However, the results do not stretch any further than that. Giving the model a more complicated input usually results in a nonsensical reply, which prevents it from having a coherent conversation with the user.

We present a few hypotheses as to why we did not get better results, with suggestions on how they could be solved. We display high hopes for future work in the area and present a few suggestions of what could be the next steps.

Keywords: chatbot, dialogue, dialogue modeling, dialogue system, artificial neural networks, recurrent neural networks, rnn, lstm, user testing, deep learning

Acknowledgements

We thank Mikael Kågebäck for the support and all the useful feedback. We also thank all the people that participated in the evaluation of our chatbot and David Benyon who provided us with the questionnaire for the user evaluation.

Viktor Anderling, Jonathan Orrö, Gothenburg, May 2016

Contents

List of Figures

List of Tables

1
Introduction

The idea behind a talking machine has been around for at least 60 years, ever since Alan Turing presented the famous Turing test [2]. This is a test to tell if a machine is smart enough to fool a human into thinking that the machine is a human as well. Today talking machines are often referred to as *dialogue systems* or *chatbots*. In 1966 Weizenbaum created ELIZA [3], which is often referred to as the first chatbot. Since then many chatbots have been created with different purposes. Some chatbots are created with a very specific purpose in mind, like automated customer support or aid in the booking of train tickets. Others are simply created to have someone to talk to for entertainment [4].

Many of these chatbots are created using some sort of rule-based system, such as AIML [5] or Lingubot [6]. In a rule-based system the creator of it must explicitly define what kind of response the chatbot should give for each possible input. This severely limits the amount of variation that a system of this kind can produce. While this is often acceptable for chatbots with a specific purpose and a known domain, it is very hard to construct a convincing one purposed for general conversation by using a rule-based system.

In recent years, there have been some great successes within the area of deep learning [7], notably with the use of recurrent neural networks (RNNs). More and more, researchers have shown interest in applying neural networks to areas such as natural language processing. In this thesis, this work is continued by using a recurrent neural network for dialogue modeling to create a general chatbot.

The benefit of having a trained dialogue system is that there are, in theory, no limits to the input or output of the system. Instead of choosing a pre-made answer based on specified inputs, the network will generate the answer dynamically. This removes the need to develop a script for the chatbot, which is a time consuming and difficult process. A trained chatbot could in addition to being easier to build, possibly also produce better results than what a hand crafted script could ever achieve. A trained chatbot could potentially be very useful for general conversation, because it would not necessarily need to know any specific information and could therefore make effective use of a trained chatbot's dynamically generated output.

Chatbots for general conversation might be of use as a complement to domain specific chatbots. Reeves et al. showed in a study that people tend to connect and talk to computers as though they were human [8], which makes the chatbots often receive out-of-domain interactions in the form of small talk. People also find a computer more helpful when they can feel that it has a personality, as shown by Bickmore et al. [9]. It could therefore be argued that out-of-domain conversation is important, even for chatbots geared for a specific purpose or domain.

The goal of this thesis is to explore the territory of dialogue modeling using recurrent neural networks and to investigate how the performance of a general model trained on a dialogue corpus compares to that of rule-based chatbots. Our focus is to create a chatbot capable of having conversations with humans and in turn make these conversations as natural as possible.

The model proposed uses the now popular sequence-to-sequence model originally proposed by Sutskever et al. in 2014 [10]. This model excels in problems that require mapping of one sequence to another. Dialogue can be simplified to a problem of this kind, where one sequence is an utterance the correct response is its mapping.

The dialogue system is evaluated using an evaluation scheme based on user testing, where 10 participants test our chatbot together with a few other chatbots.

In the end, some potential could be seen in the model. The chatbot managed to find some very common patterns in the english language, such as answering 'Hello' to greeting phrases and 'Goodbye' to valedictions. It also generated complete and correct english sentences most of the time. However, the chatbot was incapable of having coherent conversations since most of the responses given where out-of-context to the dialogue. The participants of the tests rated it as the worst chatbot and interviews with the testers showed that the conversations with the chatbot felt very random. The discussion of this thesis presents some errors that might have prevented the model to achieve satisfactory results and suggests what can be done to get an improvement for future work.

2
Related Work

Many different approaches to dialogue modeling have been made in the field. Even though the approach made in this thesis is somewhat novel, the model is heavily based on other language models. Since dialogue systems are hard to evaluate using automatic methods, which are often the preferable method in machine learning, an evaluation method based on user testing was used. This section describes a few other works highly related to this thesis.

Vinyals and Le published in 2015 a paper [11] describing a model very similar to the one proposed in this thesis, where they developed what they called a Neural Conversation Model (NCM). They used a sequence-to-sequence model trained on a movie subtitle corpus and received some intriguing results. The sample dialogues they published indicates that they had produced a basic chatbot that also has a simple knowledge base, from only training on the dialogue corpus. The model is evaluated by letting participants compare its answers to questions with answers made by Cleverbot [4] to the same questions. For every question, the participant was tasked to choose the best answer. In total there were four participants and if three or four of them agreed on one answer, that one was considered best. Using this evaluation scheme they showed that their NCM performed better than Cleverbot.

Another approach to dialogue modeling using a movie subtitle corpus was made by Ameixa et al. in 2014 [12]. They wanted to create a system for handling out-of-domain interactions for systems designed to answer questions within a specific domain. Their method was to split the subtitle corpus into utterance and response pairs and use these as their data set. When receiving a message the system would then search the data set to find the utterance that most resembles the message and simply output its corresponding response.

They evaluated their system by letting people rate answers to selected out-of-domain interactions. Even though the model was simple, their system seemed to provide reasonable answers to a majority of the utterances.

The techniques used in this thesis have been used for other purposes than dialogue modeling. Sutskever et al. used a multi-layered LSTM sequence-to-sequence model in 2014 for machine translation [10], which was primarily focused on French to English translation. They trained the model on the WMT' 14 French-English data set, which includes phrases in French and their English translation. It received a BLEU score which nearly beat what was the best score at that time.

A significant amount of the resources for of this thesis has been spent on finding an appropriate evaluation method. However, this is not the first time the need for evaluating a dialogue system has arisen.

Walker et al. developed a system in 1997 called PARADISE [13] (PARAdigm for

DIalogue system Evaluation) and is a system often cited even today. The framework is developed for performance evaluation on task oriented dialogue systems where the user has a clear task to perform when initiating the interaction. Examples of where a task oriented dialogue system could be used are train ticket bookings or customer support.

PARADISE focuses on evaluating how effective such a dialogue system is and does this by running user tests where users have a clear task to accomplish during the test. Different metrics are collected during the test, such as how many commands were used, how long it took to complete the task, how many times the dialogue system did not understand the user etc. The dialogue system is then evaluated based on these metrics. The PARADISE system is also highly focused on producing results that can then be used to compare different dialogue system to each other.

Another more recent attempt at dialogue system evaluation was made by R. Higashinaka et al. in 2015 [14]. They focused their evaluation on something they called system breakdowns, which is a point in the conversation where the dialogue system made a mistake severe enough that the conversation could not continue.

Higashinaka collected the data by asking other researchers from around the world to chat with their system, and then collected all the chatlogs. In total the study used data from about 1200 dialogues, which was manually annotated on the places where a breakdown occurred and commented on what type of breakdown it was. The evaluation would then be performed on the annotated data to identify what type of breakdowns can occur and how often.

Lastly, C. Smith et al. created in 2011 a dialogue system which they called an 'affective conversation agent', which was designed to figure out what emotions the user were feeling and use this information when speaking [15]. This dialogue system uses both speech recognition and speech synthesis, which allows the user to physically talk to it. The conversation agent also has a 3D-modeled avatar that changes its expression based on the mood it tries to conceive. To gain information on how it should express itself, a camera is used to detect the users current mood from facial expressions.

This dialogue system was evaluated in an accompanying paper by Benyon et al. [16]. Since the original paper tried to develop a natural dialogue system, the evaluation also focused on evaluating the naturalness of the conversation. They performed user testing sessions and based the evaluation on three different parts: metrics taken during the testing, answers from a questionnaire handed out to the users after the testing sessions and the chat logs themselves.

3
Background

To understand the model used for this thesis, some background knowledge in the area of neural networks and deep learning is required. This section will describe the basics of neural networks, how they are trained and the architecture in which they are used for dialogue modeling.

3.1 Neural networks

Neural networks is a type of mathematical model often used to statistically model patterns in a dataset. It is inspired by how the human brain is believed to function [17].

In the brain, *neurons* connects to each other via their synapses, which is a one way connection. These connections can be of varying strength. A neuron can fire an electrical signal, and does this when it has gotten enough stimulus from other neurons via their synapses. This signal is then transmitted to other neurons which this neuron's synapses connects to [18]. In this thesis, the terms *cell* and *neuron* will be used interchangeably. In artificial neural networks, this phenomena is commonly simplified with a model of the neuron introduced in 1943 by McCulloch and Pitts [17]. The value, or *output* of an artificial neuron x_i is updated by the values of all other neurons x_k that connects to it, where $i, k \in [1, ..., n]$ and n is the number of neurons. The output of these neurons x_k is weighted by a factor $w_{i,k}$ corresponding to the strength of the connection neuron k has to i. Additionally, each neuron i has an associated bias value b_i. A single connection to a neuron can be seen as modeling a line similar to the line equation $x_i = w_{i,k} \cdot x_k + b_i$, thus many neurons can model many lines and capture more complex patterns.

More formally, the value of a neuron i is calculated in equation 3.1, where x_k is the value of neuron k, $w_{i,k}$ is the weight from neuron k to i and b_i is the bias of neuron i. Note that a weight of 0 corresponds to those neurons effectively having no connection.

$$x_i = g\left(b_i + \sum_{k \neq i}^{n} w_{i,k} \cdot x_k\right) \tag{3.1}$$

The function g is the neuron's *activation function*, usually either tanh or the logistic function. Its purpose is to scale the value of the neuron such that it always stays inside some interval. The logistic function, denoted here as $\sigma(.)$, is defined in equation 3.2 and can take values in the range $[0, 1)$. Recall that the neurons in the brain fire a signal given enough stimulus. By using the logistic function, the output of a

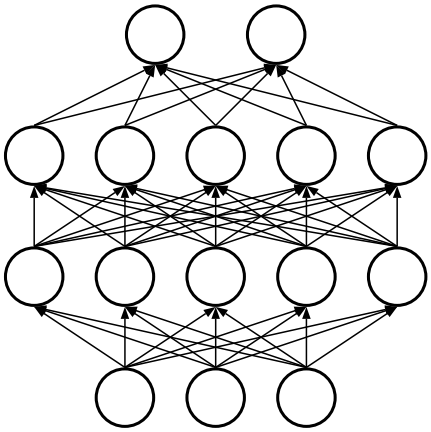

Figure 3.1: An example of a feed forward neural network. The circles are neurons and the arrows are the weighted connections. This network has three input neurons, two output neurons and two hidden layers with five neurons each.

neuron is effectively modeled by the probability that it will fire rather than being binary on or off.

$$\sigma(\alpha) = \frac{1}{1 + e^{-\alpha}} \tag{3.2}$$

If several of these neurons are connected to each other, we get a *neural network*. Some neurons will be what is called input neurons, their state being set from the start. Given some sort of starting state, the other neurons can be updated according to the rule given in equation 3.1. The idea behind a neural network is that there should exist a combination of values for the weights and biases that lets the network produce the desired output given a corresponding input. This section will describe the type of neural network used in this thesis. It will also describe how the network is trained and optimized to find the right weights and biases.

3.1.1 Feed forward neural networks

A common type of neural networks are the feed-forward type of networks. Feed-forward neural networks are categorized by how the information flows through them: it only flows forward. In other words, they are directed acyclic graphs (DAGs). Feed-forward networks are among the simplest possible networks and one of the first that was used [19]. They are comprised of layers of neurons, where neurons in one layer are connected to some or all in the next. The first layer is called the input layer, the last is the output layer and the layers in between are the hidden layers. The simplest form of feed forward network is a fully connected one, where each neuron in one layer is connected to all neurons in the next one. See figure 3.1 for an example.

Much like in the general neural network, information flows between neurons by weighted sums of the connected neurons followed by an activation function. Given a weight matrix W^l where $W^l(i,k)$ is the weight from neuron k in layer $l-1$ to neuron i in layer l, the biases $b^l(i)$ and a layer of hidden neurons h^l, the value for the i'th neuron $h^l(i)$ in layer l is calculated as shown in equation 3.3. Note that we use the logistic function as the activation function.

$$h^l(i) = \sigma\left(b(i) + \sum_k W^l(i,k) \cdot h^{l-1}(k)\right) \tag{3.3}$$

Taking all neurons of a layer into account, the neurons in the next layer h^l is calculated with the matrix multiplication in equation 3.4. Recall that h^l is a vector of neurons, thus $\sigma(.)$ will be applied element-wise on its argument.

$$h^l = \sigma\left(W^l h^{l-1} + b^l\right) \tag{3.4}$$

The input layer has no connection to it, instead its values are set manually each time the network is run. Likewise, the output layer has no connections from it, since its values are taken as the result of the network's computations. Because the network is arranged as a DAG, given the values of one layer, the values for the next layer can be updated simultaneously. Thus, by setting the values for the input layer, the following layers is calculated consecutively until the values for the output layer can be obtained.

To make the network output something meaningful, we need some means of adjusting the weights to improve the output. For this a method called *backpropagation* is used and is described in section 3.1.2.

3.1.2 Weight Optimization

We have seen how a feed forward neural network can give a corresponding output $q(x)$ given some input x if it has its weights set correctly. However, finding the correct weights is far from trivial. If each of the neurons in the output layer of the network represents different classes that we want the network to learn, we can see the output layer as a distribution of the probabilities of the classes that the network believes an input x belongs to. For example, if the network has 3 classes, given an input x the network might predict $q(x) = (0.1, 0.75, 0.15)$, meaning that it believes the second class is most probable to belong to x. Given a correct label $p(x)$, for example $(0, 1, 0)$, we can tell if the network made a correct prediction (in this case it did). We need a way to know *how* bad a certain set of weights are given the correct labels and the predicted probabilities, calling this the *entropy* of the network. We define $H(p, q)$ as the entropy, where $p(x) \in \{0, 1\}^{|x|}$ is the true probability (the correct label) and $q(x)$ is the predicted probability of data point x. For this we must define a *loss function*. One of the most common loss functions is called *cross entropy* and is defined in equation 3.5. A subset of all data points N, called mini-batch, is usually used when calculating the loss to make the training more tractable.

$$H(p, q) = -\frac{1}{N}\sum_x^N p(x) \cdot \log q(x) \tag{3.5}$$

We also need an algorithm to minimize this loss function with respect to the weights of the network. This algorithm can vary greatly depending on the task, but usually involves calculating gradients of the weights in the network. This is called *backpropagation* and is done by applying the chain rule on the derivative

of the loss function back through the layers in the network. The most common optimization method for neural networks is *Stochastic Gradient Descent* (SGD). The SGD-algorithm uses backpropagation to calculate the gradients for each weight in the network and minimizes the loss function by iteratively taking small steps in the direction of the negative gradient. An update of SGD is defined in equation 3.6, where W_t is the weights at time t of the network, $\eta > 0$ is the step size or *learning rate* and $\nabla H(p, q)$ are the gradients.

$$W_{t+1} = W_t - \eta \nabla H(p, q) \tag{3.6}$$

The learning rate η needs to be set manually, but there are more advanced optimization methods which can lessen the importance of this parameter slightly. One of these is the *Adaptive gradient algorithm* (AdaGrad) which was introduced by Duchi et al. [20]. It is an optimizer that assigns individual learning rates for each weight in the network dynamically during training. AdaGrad is especially useful for optimizing on sparse features, such as when the output classes is a bag of words. The algorithm will then give more importance to less frequently occurring features, enforcing a form of equity.

3.1.3 Recurrent Neural Networks

Regular feed forward networks cannot take input sequences of arbitrary lengths. This is required for many applications such as text processing where the length of each sentences may vary. For example, one might use each word of a sentence one by one as input. Since normal feed forward networks has no persistent state, it is not possible to detect dependencies between cohesive words being input to the network at different times. Recurrent neural networks solves this issue by introducing a *recurrent connection* from a neuron's previous state to its next one. A visualization of a single recurrent cell can be seen in figure 3.2.

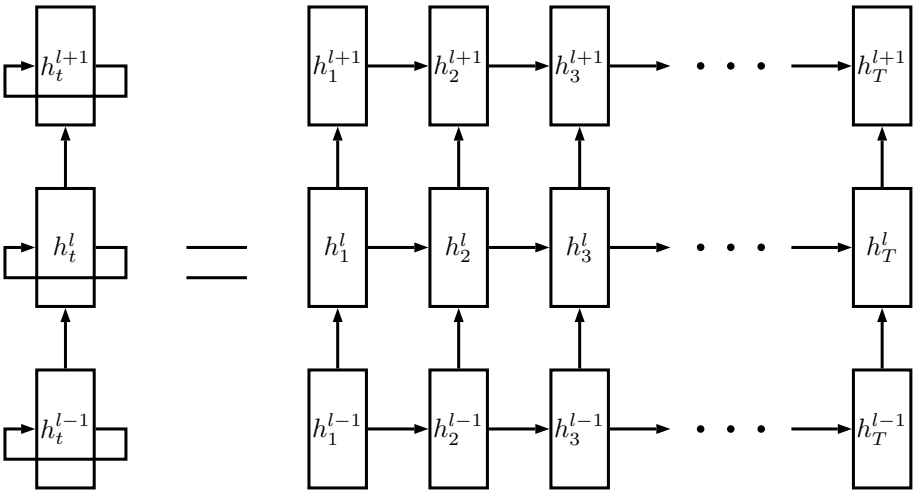

Figure 3.2: A visualization of a single recurrent cell at time t to the left, and the same cell unrolled in time to the right. Note that the cell affects both the cells in the next layer and the cells in the same layer and next time step.

We define h_t^l as the hidden neurons at layer l and time step t. To make following equations simpler, we also define linear transformation for an input or layer h of length α, a weight matrix W of size $\beta \times \alpha$ and a bias vector b of size β.

$$T_{\alpha,\beta}(h) = Wh + b \tag{3.7}$$

This transformation is equivalent and a shorthand to the equation 3.4 for a feed forward network in section 3.1.1, but without the activation function applied. For a standard recurrent neural network, h_t^l is updated according to equation 3.8, where layer $l-1$ and l contains m and n neurons respectively.

$$h_t^l = \sigma\left(T_{m,n}\left(h_t^{l-1}\right) + T_{n,n}\left(h_{t-1}^l\right)\right) \tag{3.8}$$

Note that each instance of $T_{\alpha,\beta}(.)$ has its own unique weight matrix W and bias vector b. The first term inside the logistic function in equation 3.8 contains the transformation from the previous layer $l-1$ at the current time step t, while the second term contains the transformation from the current layer l at the previous time step $t-1$, thus clearly demonstrating how the network indeed looks at itself one step back in time.

3.1.4 Long short-term memory

While the RNN has a memory because it feeds its states forward in time, in practice the network will quickly forget patterns after a couple of steps. It has been shown by Bengio et al. that it is very difficult to train a standard RNN to find patterns that exist over a large number of steps [21]. To solve this problem, an extension to the standard RNN called Long Short-Term Memory (LSTM) is often used instead.

An LSTM network is built up by LSTM cells just like an RNN is built up by McCulloch-Pitts neurons. Every cell has input and output, and is connected to itself to get the temporal aspect. Just like in RNNs, LSTM networks usually have multiple layers with many cells in each layer. Figure 3.3 shows a basic LSTM cell. This is a single cell k in layer l at time step t. In this figure the output is a single entry in the input vector to the next layer and next time step. There are several variations of LSTM, where the one described here is the original model presented by Hochreiter and Schmidhuber in 1997 [22].

The basic concept behind an LSTM cell is to use something called gates to control what information the cell will remember, forget and output. These gates allow the cell to remember information significantly longer than the neuron of a standard RNN.

Recall from section 3.1.3 that the output of layer l at time step t is defined as h_t^l. Similarly, c_t^l is defined as the memory states for layer l at time step t. The memory state acts as a persistent memory and is the main feature of the LSTM. The full equation to calculate the new memory state is described in equation 3.9.

$$c_t^l = f_t^l \odot c_{t-1}^l + i_t^l \odot g_t^l \tag{3.9}$$

Where $\odot$ is element-wise multiplication. The different parts of this equation will be explained in detail below. First, the old memory state is element-wise multiplied

by f_t. This is the forget mechanism. f_t is defined in equation 3.10 and calculates which properties of c_{t-1}^l that should be forgotten.

$$f_t^l = \sigma \left[T_{m+n,n} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} + b_f \right] \tag{3.10}$$

Recall that the size of layer $l-1$ and l is m and n, thus the transformation T is done on a vector of size $m+n$ into a vector of size n. The use of element wise logistic function guarantees that the values are between 0 and 1, where 0 would mean completely forget and 1 completely remember what is currently stored in the memory state.

Note that an additional bias term b_f is added to the forget gate, which is set to 1.0. This allows the model to store information more easily during the early stages of training as shown by Jozefowicz et al. [23].

After this information from the input and the previous output is added to the memory state by adding $i_t^l \odot g_t^l$. The input gate i_t^l decides how much of each input that should be added and g_t^l are the actual values that will be used as input to the cell. i_t^l and g_t^l are calculated in equations 3.11 and 3.12.

$$i_t^l = \sigma \left[T_{m+n,n} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} \right] \tag{3.11}$$

$$g_t^l = \tanh \left[T_{m+n,n} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} \right] \tag{3.12}$$

Note that g_t^l use the tanh function instead of the logistic function. This allows the input values to the LSTM to take on values between -1 and 1. The tanh function is basically just a translated and rescaled logistic function, as seen in the equality in equation 3.13

$$\tanh(x) = 2\sigma(2x) - 1 \tag{3.13}$$

After the memory state is updated, it needs to be decided what the cell should output. This is done with the last gate o_t, which regulates what properties of our memory state we will use in the output and is described in equation 3.14.

$$o_t^l = \sigma \left[T_{m+n,n} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} \right] \tag{3.14}$$

The output for the current layer h_t^l is calculated as shown in equation 3.15, which is sent to the next layer and time step. The tanh function is first applied to the memory state to normalize it to the interval [-1, 1] and the result is then scaled with the output gate.

$$h_t^l = o_t^l * \tanh\left(c_t^l\right) \tag{3.15}$$

In equation 3.16, lstm(.) is denoted as a function calculating the next output of an LSTM-cell according to equations 3.9 to 3.15 (The internal memory state is implicitly updated).

$$h_t^l = \text{lstm}_t^l\left(h_t^{l-1},\, h_{t-1}^l\right) \tag{3.16}$$

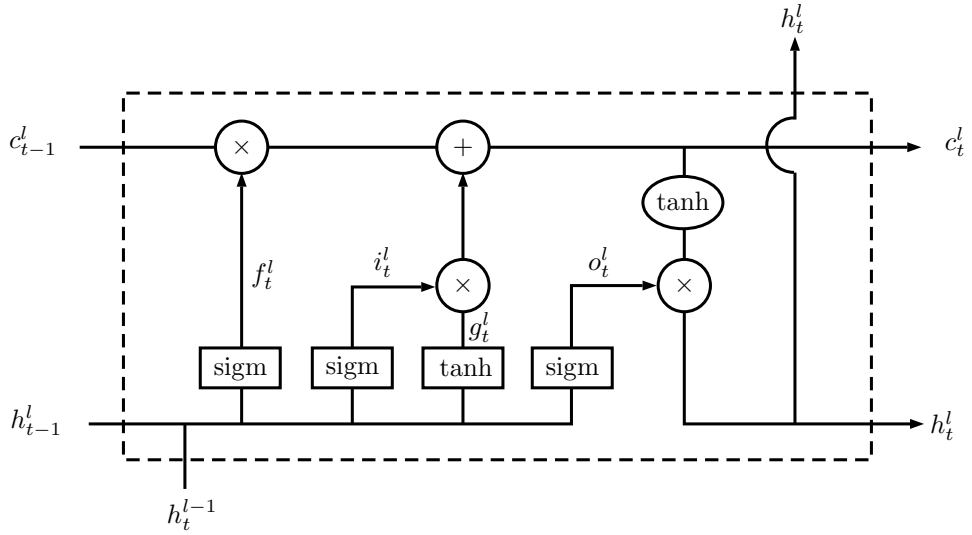

Figure 3.3: A layer of LSTM-cells at layer l and time step t. The ellipses denote element wise application of the operation inside it. The boxes is an application of the transform T defined in section 3.1.3 followed by the activation function in the box. Merging lines means concatenation of the vectors.

3.2 Tokenizing

When training a neural network on a word to word basis, it is not very practical to use the actual words as input. The interest does not lie in how the word is constructed, only what word it is. Therefore the text is *tokenized* before it is passed on to the network, which simply means that every unique word is replaced by an identifying number.

A vocabulary is created by finding each unique word that occurs in the chosen data set (corpus) and adding them to an indexed list. Having a vocabulary that covers all possible words is usually both unnecessary and unfeasible, since a larger vocabulary will increase computation of the losses/gradients (see section 3.1.2). Instead, it is common to use a subset of the full vocabulary, containing only some of the most commonly occurring words.

Because some of the more unusual words in the corpus will appear in the training data, but not in the vocabulary, there must be some way of representing them. The simplest way to solve this is by replacing all occurrences of an out-of-vocabulary word with a special token that is treated by the network as a single word. Every instance of out-of-vocabulary words would then be treated as though they were the same words.

It is common to represent the input to the network as a *one-hot vector*. A permutation of this vector represents a word in the input vocabulary. This vector is the same length as the size of the vocabulary used, and all elements in the vector except for one is set to 0. A one-hot vector having its element at index i set to 1 will thus represent the word at index i in the input vocabulary.

3.3 Word embedding

As mentioned in section 3.2, one-hot vectors are convenient to use as input due to their simple bag-of-words-like structure. However, when training a neural network on this kind of input, the network will not be able to understand semantic similarities between words in the vocabulary, since the inputs are basically just represented by their indices in the vocabulary. This means that not only will the network need to be trained for its intended purpose, but it will also need to be trained to understand how each word correlates to each other. To avoid this a process called *embedding* is used to translate the one-hot-vector into a lower dimensional vector that contains semantic information about the word.

Usually, one layer in front of the network is designated to linearly project the one-hot input vector onto its embedding. For example, consider a one-hot input vector v and an *embedding matrix* E. The projected input v' is then calculated by the matrix product of v and E, such that $v' = Ev$. The embedding matrix E is trained the same way as the rest of the network. While it is not unusual that the size of the vocabulary, and thus the size of v, is around 100 000, the size of v' is usually well below 1000. This shows that the embedded word is a much more compact representation than the sparse one-hot vector.

The idea is that each row E_i, which we call a *word vector*, in E should represent some kind of semantic meaning of word i in the vocabulary. This semantic meaning should be invariant to any model which one would want to apply it to. Therefore, it is reasonable to assume that these word vectors can be pre-computed and inserted into the model from the start of a training. This will allow the network to achieve better results earlier, which can reduce training time significantly.

One method of training these word vectors is by using GloVe [24], which is an unsupervised learning algorithm that analyzes the co-occurence of words in a corpus. The result of the algorithm is a high-dimensional vector for each word, which can be used as pre-trained word embeddings for a neural network. The GloVe vectors also has interesting Euclidian properties, where similar words will appear close to each other in vector space. The closest five neighbours of the word frog, for example, are: frogs, toad, litoria, leptodactylidae and rana [1]; where litoria and rana are genuses of frogs and leptodactylidae is a family of frogs. Another important feature of GloVe vectors is that the vector difference between two words describes the semantic difference between these two words. This means that the vector difference between, for example, man and woman are roughly the same as the difference between king and queen. See figure 3.4 for more examples on this feature.

Another advantage to pre-trained word vectors is that these only needs to be trained once, and can then be copied and used for training of subsequent models. Thus, it can be trained with a substantially larger corpus than the one used for training the model.

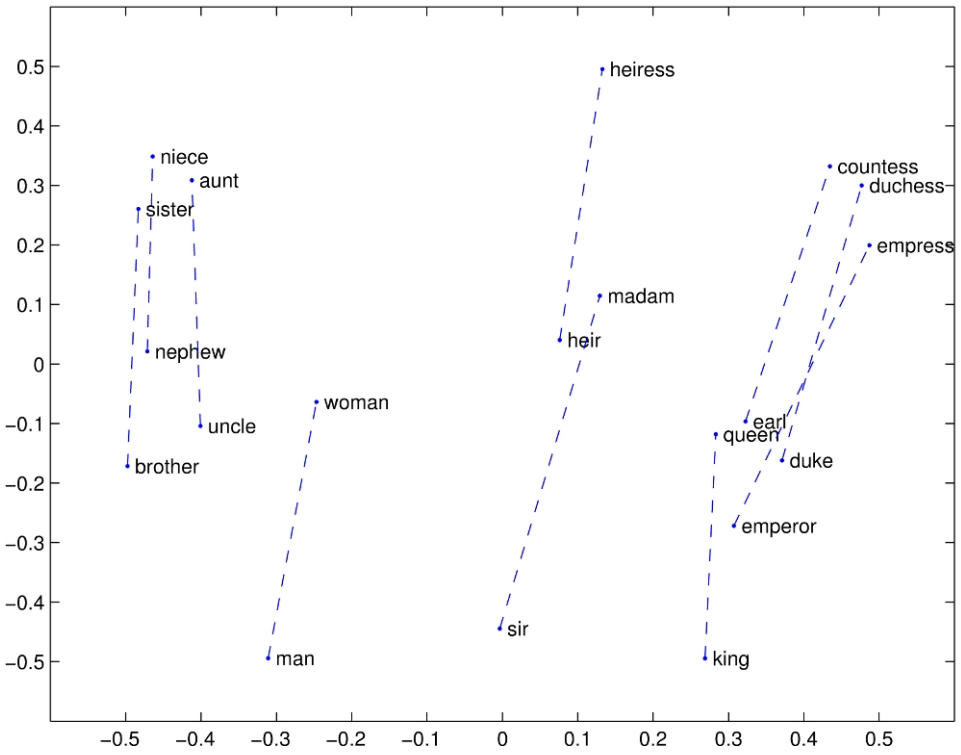

Figure 3.4: The vector difference between the words describes semantic differences. All the vector differences between these words are roughly the same, since they all describe a man and woman of a certain type. Image taken from [1].

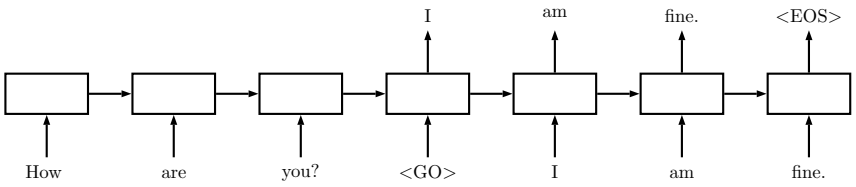

Figure 3.5: A sequence-to-sequence model which encodes the sentence "*How are you?*" and produces during decoding the sentence "*I am fine. <EOS>*". When decoding, the previously generated output is used as input for the next time step, except for the first word, where *<GO>* is used as input. The decoder stops when an *<EOS>* is generated.

3.4 Sequence-to-sequence

An increasingly popular technique for using RNNs with natural language processing is *sequence-to-sequence* (seq2seq). It is a generative neural network model which given a string of inputs produces a string of outputs, both of arbitrary lengths.

A sequence-to-sequence model is made of a recurrent neural network with L layers and is divided into an encoder and a decoder. The decoder usually has an attention model attached to it, which is described in section 3.4.1. The input is a tokenized sentence with length T, which is converted into a string of one-hot vectors $(x_1, x_2, ..., x_T)$. At each time step, a word x_t is embedded into a vector $x'_t = E \cdot x_t$, where E is the embedding matrix of size $d \times |V|$, where d is the number of embedding dimensions and $|V|$ is the size of the vocabulary. The embedded word x'_i is then fed as the input to the encoder, which consists of a multi-layered recurrent neural network with LSTM-cells.

In the first layer of the encoder at time step t, the input is set to the embedded word x'_t. The first layer is calculated according to equation 3.16 in section 3.1.4, such that $h_t^1 = \text{lstm}\left(x'_t, h_{t-1}^1\right)$ and following layers $h_t^l = \text{lstm}\left(h_t^{l-1}, h_{t-1}^l\right)$. When calculating the LSTM during the first time step where $t = 1$, an initial state h_0^l is used. Thus, the first LSTM update for all layers becomes $h_1^l = \text{lstm}\left(h_1^{l-1}, h_0^l\right)$.

When all T inputs has been fed to the encoder, the network will be used for decoding. The idea is that the last state h_T for the network should contain semantic information about the whole input sentence. h_T is kept and used as the initial state for the decoder. At each time step $t > T$, the network will output a word w_t. The network will be run until a special *end-of-sentence* symbol is produced.

The input to the decoder x_t at time $t > T$ consists of the last word w_{t-1} that the network generated, where the first decoder input is a special *GO* symbol. The predicted word w_t at decoding time t is calculated in equation 3.17, where $V(i_w)$ is the i'th word in the vocabulary and O_t is the output of probability distributions with length $|V|$. An example run of a sequence-to-sequence network can be seen in figure 3.5. The string *"How are you?"* are fed one word at a time to the network, and it will generate words until it generates a special EOS token.

$$\begin{aligned} i_w &= \text{argmax}\,(O_t) \\ w_t &= V(i_w) \end{aligned} \tag{3.17}$$

3.4.1 Attention

Attention is a technique that allows a model to focus on different parts of the input during different times of inference. When generating matching sequences, some words might be more relevant than others during different stages, especially if the input sentence contains different kinds of information.

The following attention model is based on the one described by Vinyals et al. [25]. The attention model is shown in equation 3.18, where d_t^L is hidden state of the last layer in the decoder at time t and h_i^L is the i'th hidden state at the last layer of the encoder, which represents the i'th word of the encoder input where $i = 1, ..., T$. The vector v and matrices W_1 and W_2 are learnable parameters, which means that they are backpropagated and optimized along with the other parameters (weights) of the network. If the encoder and decoder hidden states in the model are of the same size, W_1 and W_2 will be square matrices. The mask a_i^t represents how much attention to be put on encoder input word i, and the final attention a_t' is calculated as a weighted sum of all encoder states.

$$\begin{aligned} u_i^t &= v^T tanh(W_1' h_i^L + W_2' d_t^L) \\ a_i^t &= \sigma(u_i^t) \\ a_t' &= \sum_{i=1}^{T} a_i^t h_i^L \end{aligned} \tag{3.18}$$

The attention vector a_t' is concatenated with the embedded decoder input and linearly transformed into y_t' to have the correct input size. The embedded y_t can use the same embedding matrix as the encoder. The attention-affected y_t' is then fed as input into the decoder, as shown in equation 3.19.

$$y_t' = T_{(n+d),d} \begin{pmatrix} a_t' \\ y_t \end{pmatrix} \tag{3.19}$$

The decoder uses y_t' as the input to the first layers, and the hidden state of the last layer d_t^L is combined with the attention calculated with the new decoder state d_{t+1}^L to form the output distribution O_t, as seen in equation 3.20

$$O_t = T_{2n,|V|} \begin{pmatrix} a_{t+1}' \\ d_t^L \end{pmatrix} \tag{3.20}$$

3.5 Hyperparameter optimization

Neural networks can be seen as a function that takes some input and produces an output. This could of course be a very complicated function, with hundreds or thousands of parameters. Every weight in the network would, for example, be a parameter in this function. The reason the network is then trained is to find the values for these parameters that will enable the network to do the task it is designed to do.

There are however other parameters in a network that are not modified when training the network. An example would be the number of layers there are in a network, or how many neurons are in each layer. These parameters are meta-parameters and will not directly affect how the network performs; rather, they will affect the capacity of the network and how well the network trains. Even if a network has perfect meta-parameters it still needs to be trained to perform well. On the contrary, a perfectly trained network will still be limited by its size. These meta-parameters are usually called hyper-parameters.

Hyper-parameters needs to be manually set before training and it is important that they are set to good values. Finding these values is however not a trivial task, though there are a few techniques that are commonly used. One of them is called Grid Search.

When using grid search one first defines the hyper-parameters that a search is performed on and the search space for each parameter. These search spaces are simply sets of possible values. The network is then trained on all different combinations of values, creating different models. Their performance is evaluated and the best setting of hyper-parameters can then be determined. A setting is evaluated by running the network with the selected hyper-parameter values with evaluation data, which is separate from the training data. The average error can be computed using a loss function such as cross entropy (3.5), or some other performance test.

For example, given two hyper-parameters, number of layers L and number of neurons in each layer N, a search space can be defined as seen in equation 3.21.

$$\begin{aligned} N &= \{1400, 1600, 1800, 2000\} \\ L &= \{2, 3, 4, 5\} \end{aligned} \tag{3.21}$$

The next step would be to train the network with every possible configuration of these values. When the network has finished training, the network is run with evaluation data and the average loss is calculated. The settings that produces the lowest loss will be the best configuration.

3.6 Regularization

When training large models with many variables on a complex data set they are very prone to overfitting. Overfitting is when the learning model detects patterns in details of the data that are not general patterns. An overfitted model tends to be bad at predicting and generalizing on unseen data, see figure 3.6 for an example. In the figure a line has been fitted to the data to separate the red dots from the blue crosses. If a new point is added and had to be labeled by the model, the one to the left would probably perform much better than the model to the right. The model to the right has been fitted too hard on the data.

Regularization methods are the main way to prevent overfitting. The regularization methods used in this thesis are described below.

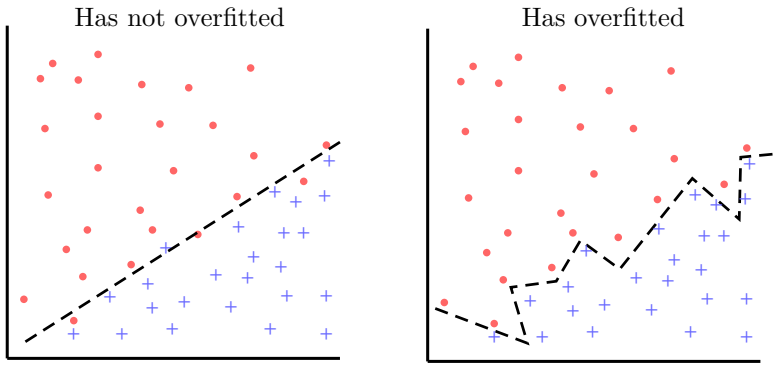

Figure 3.6: Figure of two different networks trained on classifying crosses and dots in a 2-D space. The dotted line represents the classification boundary that the network has learned. The network to the left has learned the true classification boundary, while the network to the right has overfitted, which will cause it to perform worse on data not in the training set.

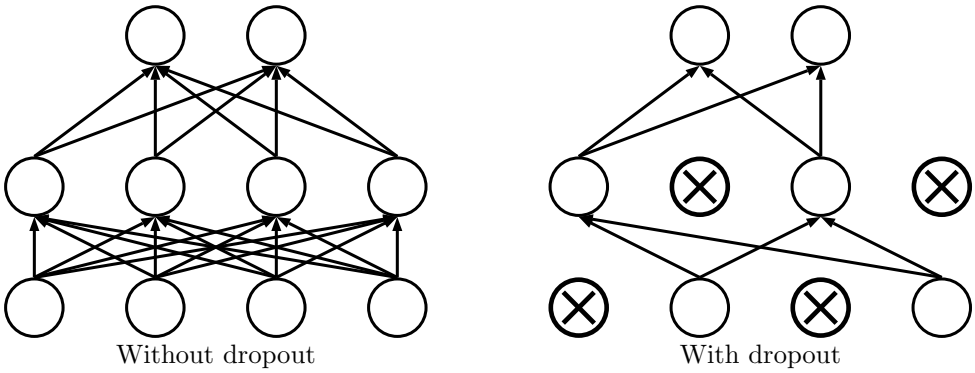

Figure 3.7: To the left is a network without dropout and to the right is an example of the same network with dropout. Some neurons have randomly been dropped, shown by an X in them.

3.6.1 Dropout

Dropout is a widely used regularization method developed by Srivistava et al. in 2014 [26]. The method is very simple. During training, neurons are dropped temporarily with a certain probability. The dropped neurons will not produce any output and will not affect neurons connected to it. The intuition behind the idea is that units will not co-adapt and in turn will decrease overfitting. Srivistava et al. proposes that a dropout probability of 50% is near optimal for most networks and shows empirically that networks that uses dropout outperforms networks that does not use them in many different areas. See figure 3.7 for en example of a network using dropout.

3.6.2 Gradient clipping

Gradient clipping is aimed at combating the exploding gradient problem that often occurs in recurrent neural networks trained with the backpropagation algorithm.

The problem, which was described in detail by Bengio et al. in 1994 [21], depicts how during training the gradients for certain weights in the network tend to grow exponentially in size, which negatively affects the performance of the network.

One way to avoid it is to perform gradient norm clipping, a method developed by Pascanu et al. in 2012 [27]. This is done by clipping all the gradients if some of them are too large. Gradient clipping is formalized in the algorithm below, where $||\hat{\boldsymbol{x}}||$ is the l^2 norm of $\hat{\boldsymbol{x}}$.

Calculate gradients and store in $\hat{g}$;
if $||\hat{g}|| >= threshold$ **then**
  $\hat{g} \leftarrow \frac{threshold}{||\hat{g}||}\hat{g}$
end

$$||\hat{\boldsymbol{x}}|| = \sqrt{\sum_{x \in \hat{\boldsymbol{x}}} x^2}$$

When using gradient clipping, the *threshold* is an additional hyper parameter in the model that needs to be set before training.

3.6.3 Dropword

Dropword is a regularization technique rather similar to dropout and is also very simple. It was proposed by Kågebäck and Salomonsson in 2016 [28], where they trained a network to learn the meaning of words in a text by looking at the words textual context. To decrease the dependency the network has on individual words, random words would be dropped and replaced by a special tag.

4
Model details

Two slightly different sequence-to-sequence models were trained separately. In one model the state of the network was reset between each line and in the other it was preserved between lines during a conversation. In this section both of these models are described.

4.1 State-resetting model

The state-resetting model is a sequence-to-sequence model based on the one presented by Vinyals et al. [25]. Sequence-to-sequence is described in more detail in section 3.4. Before testing, the model was trained for 14000 time steps with a batch size of 64 using the AdaGrad optimizer. A single neural network with 2 layers and 1700 LSTM cells was used for both encoding and decoding. 90000 of the most commonly occurring words were used for the vocabulary. For words not in the vocabulary, a special *UNK* token was used. Gradient clipping was applied with a norm threshold of 5. Dropout with probability 0.5 was applied to all non-recurrent connections. A similar method to dropword described in section 3.6.3 was also used, though only punctuation was dropped. This was motivated by seeing early during training that the model fitted very strongly to the type of punctuation that appeared in the input. The dropword applied here could potentially make the model be less dependent on whether a sentence ended with a question mark or not, and more on the actual content of the sentence.

The model used pre-trained GloVe vectors [24] with 300 dimensions as initial word embeddings (see section 3.3). Since the GloVe embeddings E_{GloVe} were not trained on the data set used for training the model, about 10% of the words in the vocabulary did not have an embedding in E_{GloVe}. For each word $(v_i, ..., v_j)$ in the vocabulary, including special tokens, which did not already have an embedding in E_{GloVe}, new embeddings $(e_i, ..., e_j)$ were generated and appended to E_{GloVe} such that a new embedding matrix $E = E_{GloVe}||(e_i, ..., e_j)$ was acquired. These new embeddings were randomly initialized with the same per-dimension mean and variance as the pre-trained embeddings in E_{GloVe}. The new embedding matrix E is then jointly trained with the network.

When training and testing the first sequence-to-sequence model, the initial memory state and hidden state for the first word in each sentence was set to all zeroes such that:

$$\begin{aligned} h_0 &= [0,\ 0,\ ...,\ 0,\ 0] \\ c_0 &= [0,\ 0,\ ...,\ 0,\ 0] \end{aligned} \tag{4.1}$$

During user testing, which is described in section 6, noise was applied to the output distribution of the model. It was observed that the network would otherwise produce answers with very little variation. The noise was applied at each time step to produce O'_t as shown in equation 4.2, where O_t is the predicted probability distribution of words in V at time t, $\gamma \in [0, 1)$ is the noise level and R is a vector of uniformly randomized numbers such that $\forall r \in R; r \in [0, 1)$. During the user testing, the noise level γ was set to be 0.2.

$$O'_t = (O_t - \min(O_t)) \otimes (1 - R \cdot \gamma) \tag{4.2}$$

4.2 State-preserving model

The second sequence-to-sequence model uses the same parameters as the first one. The difference from the state-resetting model is that here the memory state and hidden state of the network is preserved between sentences in the same conversation. See section 5.2 on how the conversations were extracted from our data set. For every conversation (x_1, x_2, x_3, ..., x_{s-1}, x_s), where x_i is the i:th sentence in the conversation and s is the number of lines in the conversation, training-data points consisting of an input and a target were created on the form $\{x_i, x_{i+1}\}$, such that a conversation of inputs is ($\{x_1, x_2\}$, $\{x_3, x_4\}$, ..., $\{x_{s-1}, x_s\}$). When the model has been fed $\{x_i, x_{i+1}\}$ and generated an *EOS*-token at time t, the states c_t and h_t for the network is kept and used as the initial state for the data point $\{x_{i+2}, x_{i+3}\}$, such that $h_0 := h_t$ and $c_0 := c_t$. When the previous conversation ends and a data point $\{x_1, x_2\}$ from new conversation is used, the states are initialized to zeroes as shown in equation 4.1.

4.3 Hyper parameter optimization

Hyper parameter optimization was performed on our network as described in section 3.5. The only parameters that were optimized was the layer size and the number of layers.

The search space used for the layer size was $1100, 1300, 1500, 1700$ and number of layers was $2, 3, 4, 5$. Since both of these parameters affected memory usage and there was a limit to how much memory that could be used when performing the training, not all of the possible combinations of values could be tested. It was found that the trade-off of using larger and fewer layers gave the best performance, thus two layers of 1700 neurons each were used.

5
Data sets

During training the OpenSubtitles data set [29] were used. This is a data set consisting of movie subtitles from thousands of movies and TV series taken from the OpenSubtitles website [30]. The data set is available as a plain text version, which is a single text file of every subtitle available where every line in the text file is a line uttered in a movie. It is also available in XML format that contains additional meta data about the timing of the lines in a movie. Every line is accompanied by starting and ending time stamps and the data set is split up into the movies they belong to. Both versions of the data set were used.

The data sets were tokenized before training (see section 3.2) and put into utterance-response pairs, where one line would be the utterance and the line following it would be its response.

The state resetting model was trained using the plain text version of the data set, since the model did not require any more information than the actual lines. The state preserving model however required a data set split up into conversations. To do this, the time stamps included in the XML version of the data set were used. This section will describe how the different data sets were prepared and used during training.

5.1 Plain text version

For the state resetting model, the plain text version of the OpenSubtitles data set was used. Since there are no information in this data set from what movie, scene or even character a line is originating, an assumption was made that every other line is a response to the one before it. The data set was not perfectly formatted, so some pre-processing had to be made to remove noise. An example of how the plain text version is formatted can be seen in figure 5.1.

5.2 Conversations data set

The state preserving model needed a data set separated into conversations to know when to preserve the network's state and when to reset it. To be able to split the data set the XML-version of the OpenSubtitles data set was used. The XML-version has extra meta data about the scripts such as time stamps for when every line is spoken in a movie. These time stamps are important meta data which was used to split the data set into conversations. Figure 5.2 shows an example of how the XML data set was formatted.

I did not hit her, I did not... Oh, hi Mark!
Oh, hey Johnny, what's up?
I have a problem with Felicia, she said I hit her.
What? Well, did you?
No, I did not!

Figure 5.1: An example on how lines are formatted in the plain text version of the OpenSubtitles dataset. No additional information is given about the lines, other than their order.

An assumption was made that if two lines were spoken with a short time between them, they would be part of the same conversation. On the contrary, if two lines were spoken with a long enough delay, they are part of different conversations. This of course is not a perfect assumption, since movies can cut quickly between different scenes or several conversations can be going on in parallel.

Following this assumption, there should be some value δt_{max} where, if the time between two consecutive lines is less than or equal to δt_{max}, they are part of the same conversation. However, if the time between them is larger than δt_{max} they should be the last and the first line of two different conversations.

To help find a good value for δt_{max}, the time between consecutive lines was plotted from several movies. An example of such a graph, showing only the lines of one movie can be seen in figure 5.3. Every point in the graph is a line spoken in the movie, where its numerical index in the movie is shown along the horizontal axis and its corresponding value along the vertical axis is how many seconds ago the last line was spoken. The lines varies in height which means that time between spoken lines varies. A clear pattern can be seen, where a tall line is often followed by many small. Given the original assumption, this would mean that a tall line is the start of a new conversation and all the small lines following it are part of this conversation. Using this information, a reasonable value for δt_{max} could be found more easily.

If δt_{max} is chosen to be too large, conversations that do not belong together will be grouped together. This will introduce noise to the model when a new conversation will start with the memory of old conversations. If δt_{max} is chosen to be too small, conversations will be fragmented, resulting in not being able to remember things said in a conversation since the state will be reset in the middle of it. In the end, it was deemed safer to make an over estimate compared to an under estimate. A value of 25 seconds was used for δt_{max} when generating the data.

```
<s id="78">
    <time id="T110S" value="00:06:37,090" />
    <w id="78.1">It</w>
    <w id="78.2">says</w>
    <w id="78.3">that</w>
    <w id="78.4">.</w>
    <time id="T110E" value="00:06:39,024" />
</s>
<s id="79">
    <time id="T111S" value="00:06:39,092" />
    <w id="79.1">The</w>
    <w id="79.2">system</w>
    <w id="79.3">,</w>
    <w id="79.4">it</w>
    <w id="79.5">'s</w>
    <w id="79.6">a</w>
    <w id="79.7">monarchy</w>
    <w id="79.8">.</w>
    <time id="T111E" value="00:06:40,616" />
</s>
```

Figure 5.2: An example on how lines are formatted in the XML version of the OpenSubtitles dataset. Every line has a start time stamp and an end time stamp.

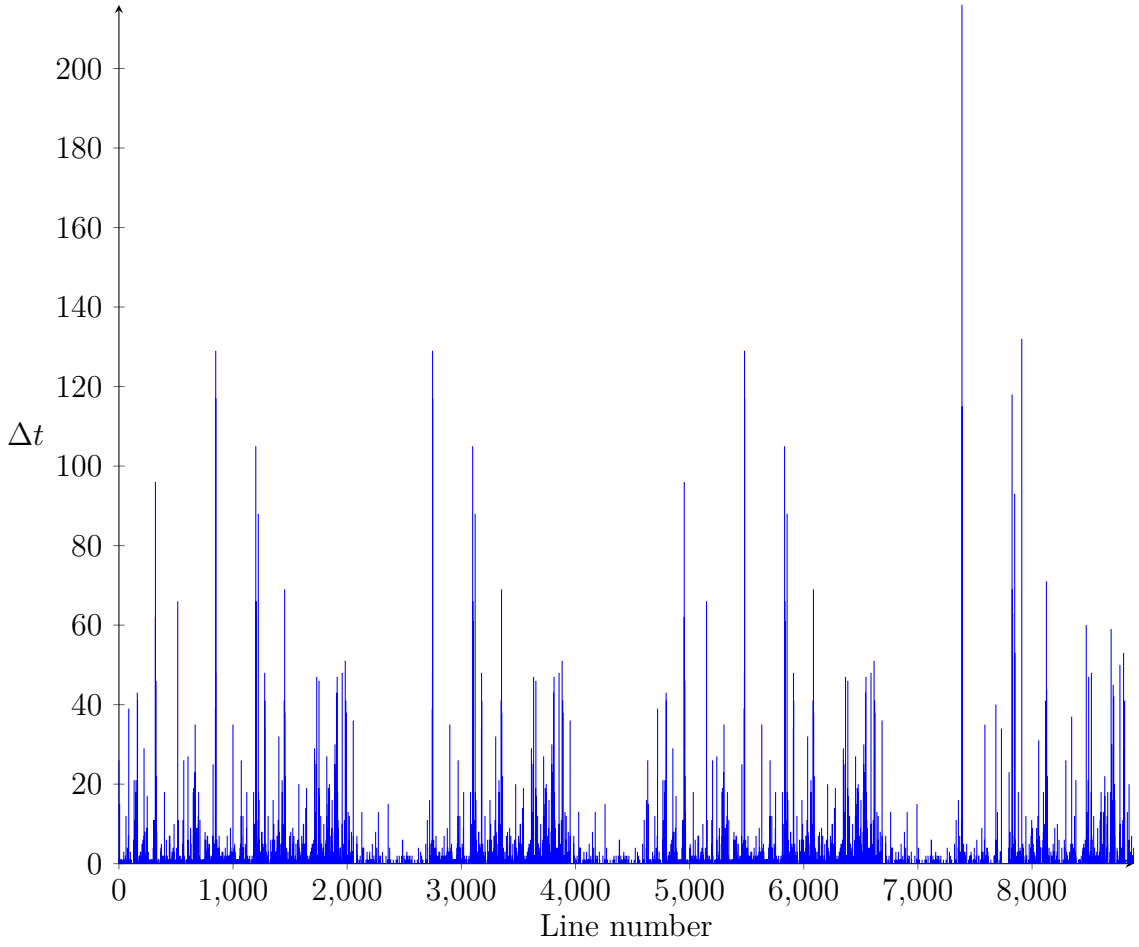

Figure 5.3: This graph shows the time between lines from a single movie in our data set where each bar corresponds to one line. The line number on the horizontal axis denotes its chronological index in the movie and Δt denotes the time in seconds between itself and the previous line.

6
Evaluation

When creating a dialogue system focused on general conversation, there is a need to find a way to evaluate how good or 'natural' the conversation it produces is. Evaluating how natural a dialogue feels is not a trivial task. It is hard to even define what natural means. Optimally, there would be a way to computationally analyze the results and get a numerical score of how the dialogue system performs. This score could then easily be compared to other dialogue systems. Since creating an evaluation method of this kind would be a research topic by itself, user testing is used instead to evaluate the results.

The user evaluation method used in this thesis is based on the method described by Benyon et al. [16]. The principle of the test is to let participants test the dialogue system and give their impressions on how the conversation felt via a questionnaire and an interview.

Since no well known method to evaluate the chatbot is used, there is no data to compare the results to. Therefore, data is collected by testing other chatbots in the same manner. A human, which the participants are made to believe is another chatbot, is also evaluated and is used as a golden standard for how a conversation companion should behave. In total, 4 different chatbots and a human are tested. During the test the state-preserving model, state-resetting model, ALICE [5], Cleverbot [4] and the human are evaluated. The participants are not given information beforehand on the details or ordering of the chatbots and they are not informed that one of them is a human.

6.1 Test procedure

The test was done physically on location at Chalmers University. The participant used a computer during the whole test and was directed to two webpages, one with the chat client that the participant used to talk to the chatbots and the other containing questionnaires the participant answered after every chatbot.

Since it was important that the participant did not know what chatbot it was talking to, a webbased chat client was used that communicated with a server that ran the chatbots. The same chat client could then be used for all chatbots, so there were no clues for the participant to know what it was talking to.

The test began by letting the participant talk to the first chatbot, which in all tests was ALICE. They were given instructions to pretend that the chatbot was human and to try and connect with it. The participant was given 5 minutes to talk with the chatbot about any subject. After the time was up, the participant

The conversation was coherent
The Chatbot demonstrated emotion at times
The conversation between myself and the Chatbot felt natural
I thought the Chatbot remembered earlier parts of the conversation

Figure 6.1: An excerpt of four examples from a total of 26 statements which was used in the user evaluation

answered a questionnaire and a short interview was held where the participant was asked to rate the chatbots in relation to each other and to come with any feedback or comments.

6.2 Questionnaire

The questionnaire was the primary evaluation tool. It was answered directly after the participant had held a conversation with a chatbot, and was filled in once for every chatbot that was tested. It is made up of 26 statements about the quality of the conversation and the participant was tasked to rate these statements on a five point scale from "Strongly disagree" to "Strongly agree". See Figure 6.1 for examples on statements from the questionnaire. The full questionnaire is available in appendix A.

The statements used in the questionnaire is a subset of the 33 statements developed by Benyon et al. [16]. Since they evaluate a different type of chatbot in their paper, only 26 of the statements were deemed relevant.

These statements were developed around six different themes empirically shown to be significant for successfully creating human connection with a digital companion [31]. The themes are naturalness, utility, participant-companion relationship, emotion demonstrated, personality and social attitudes.

6.3 Participants and Data

The user test had 10 participants in total. The ages ranged from 23 to 61, with an average age of 33. Most of the participants were students from either Chalmers University or the University of Gothenburg. All of the participants spoke english as a second language, but most rated their ability to speak and understand it as very good. Answers to the questionnaire, chat logs from the chat sessions and notes taken by us during the interviews was collected and used in the evaluation.

7

Results and discussion

In this section we will first give a description of how the final model behaves when talked to and then present the results from the user evaluation. We will also discuss the model performance and what might be done to improve it. Finally, we present a few ideas for further work.

7.1 General behaviour of the chatbot

The final model is unable to have a continuous conversation, and this is due to the fact that its replies often feel nonsensical. Most of the time the chatbot gives answers that are out of context and does not convey any sense of understanding of what the user is saying. See figure 7.1 for a typical example of what a typical conversation looks like. The dialogue system will not stick to a topic, instead saying something different after every reply and when asked a direct question it rarely answers it. When the question "Are roses green?" is asked to the system, it replies with "I can't see you.", an answer that would only be appropriate in a very specific context.

Even though the chatbot did not perform well as a conversation agent, we did however see some results. It has found some patterns from the dialogue, most notably that a greeting should be answered by another greeting and the same for valedictions. When a user says 'hello', 'hi' or another greeting it most often answers appropriately. The same is true for valedictions such as 'goodbye' or 'farewell'. The chatbot also learned to speak nearly perfect english. It is rare that it uses incorrect grammar, though happens sometimes. It also uses correct punctuation, always terminating sentences with a period, an exclamation mark or a question mark. This does show that there are some patterns, although possibly small, to find in dialogue with the relatively simple model used in this thesis.

7.2 User evaluation

Two different models were developed during the thesis work and both models were evaluated side by side. How the evaluation was done is described in section 6. To have something to compare our models to we also tested a human and two other chatbots: Alice [5] and Cleverbot [4]. The testing gave us two main sets of data to evaluate the model from: interviews and answers to the questionnaire.

7.2.1 Questionnaire answers

The answers to the statements in the questionnaire was converted to numerical values, where -2 would be "Strongly disagree" and 2 would be "Strongly agree". The average score for each chatbot was calculated for every statement. See table 7.1 for some examples of scores for a few selected statements. We cannot formulate a single score from this data for every chatbot in a meaningful way, but we can look at the scores we got on individual statements and evaluate the results from there.

Looking at the scores it is clear that both our models performed poorly. The models scored consistently worse than every other chatbot on all of the statements and it did not get a positive score on any of the statements except for 'I found the chatbot boring'. As we can see in table 7.1, even on statements such as 'I found the chatbot entertaining', where all of the other chatbots got a good score, our models got a negative score. Both of our models got scores close to each other, most probably because both of them acted in a very similar manner.

7.2.2 Interviews

At the end of the test we performed a short interview with the participant. We asked the participants to rank the chatbots on the quality of the dialogue, from the most natural to the least. Every participant rated one of our two models as the worst of them all. There were no preference to one of the two models, each of them were ranked last approximately 50% of the time.

We also asked the participants to list the chatbots they could consider being used as a conversation companion. The human and Cleverbot was listed by every participant, ALICE was picked by three of the participants but none of our models where listed by any participant. When asked why they would not consider any of our models, the same reasons were given by everyone: that the conversation felt completely random and that the they did not seem to listen to the participant.

The participants were also asked to compare our two models with each other. Most answered that they did not notice much difference, that they both felt like the same chatbot. Some of them commented that one felt slightly better than the other, but there were no consistency in which was ranked better than the other. This is probably due to on of the models getting 'lucky' and giving slightly better answers than the other for this particular participant.

7.3 Model performance

We have some ideas as to why the network did not perform as well as we had initially hoped. However, these ideas are mostly speculations based on intuition.

A difficulty when assessing the results is how much one can trust the evaluation error during training (see figure 7.2). Using cross entropy is not necessarily the optimal loss function for evaluating our network during training. For each input sentence, there are numerous possible outputs that are just as valid as the provided target sentence. This means that we may see a false negative error on whether the model performance is increasing or not. While it will still be possible to see if and

user: hi!
chatbot: hello.
user: how are you?
chatbot: you've been so much
user: i've been so much?
chatbot: i'm not going to do that, i mean, you're a fool.
user: i am not a fool!
chatbot: it's a little thing.
user: ok.
chatbot: i don't know.
user: what is your name?
chatbot: you're not going.
user: what are you doing?
chatbot: i've had to ask you.
user: what do you want to ask me?
chatbot: i don't know why you're a fool.
user: i do not think i am a fool.
chatbot: i'm not going to be a man.
user: ok.
chatbot: that's the idea that's the best.
user: are roses red?
chatbot: yes.
user: are roses green?
chatbot: i can't see you.
user: are apples red?
chatbot: i'm sorry, scarlett.
user: i have to leave now!
chatbot: i don't know.
user: goodbye.
chatbot: goodbye, dear.

Figure 7.1: An example of some inputs and responses from a chatbot trained with neural networks. We can see that its responses are nonsensical. The chatbot does however produce complete and correct sentences and not just random words.

Statement	Alice	Cleverbot	Human	First model	Second model
The conversation between myself and the Chatbot felt natural	-0.6	0.2	1.3	-1.7	-1.0
The Chatbot's responses were always appropriate	-1.0	-0.8	1.3	-1.4	-1.1
The Chatbot demonstrated emotion at times	0.6	0.6	0.0	-0.7	-0.2
The conversation was coherent	-0.1	-0.2	1.3	-1.3	-1.4
I found the Chatbot entertaining	0.9	1.3	0.8	0.0	-0.3

Table 7.1: Selected statements from the questionnaire and a corresponding average score for each tested chatbot. The score ranges between -2 and 2.

when the model overfits, the effect of this may lead to belief that the model has converged while in reality it is still improving.

7.3.1 Training time

The simplest hypotheses about the model performance is that the network did not train for long enough. A network reaches a point where it performs optimally after a certain amount of training, and we believe that our models were far from reaching that point.

Both of the models were trained for 14 000 steps at a batch size of 64, which means that they each saw 900 000 data points in total.

If we look at the training losses and the evaluation error from the training we can see that neither of them has converged (see figure 7.2, because both are still going down. A network that has not yet converged is a sign that it can still learn more about the data. However, this does not necessarily mean that the network will perform better if left to train for longer, though it is a good indicator of it.

7.3.2 Choice of hyper parameters

Many hyper-parameters were fixed during the hyper-parameter optimization and were therefore not optimized. That includes the learning rate, the dropout rate and the dropword rate. These parameters were instead guessed by intuition, and thus there is no guarantee that the values for these are optimal. They are somewhat important for both how fast the network learns and how well it performs when it is done training, so optimizing these could have improved our results.

The size of our network could also have been an issue. In general, the size of a network dictates how advanced rules it can learn, therefore it could be that our network was not large enough to be able to learn enough interesting rules to model dialogue of good quality. Józéfowicz et al. recently did an exploration into the limits in natural language modeling with RNNs and used models as large as 8000 cells per

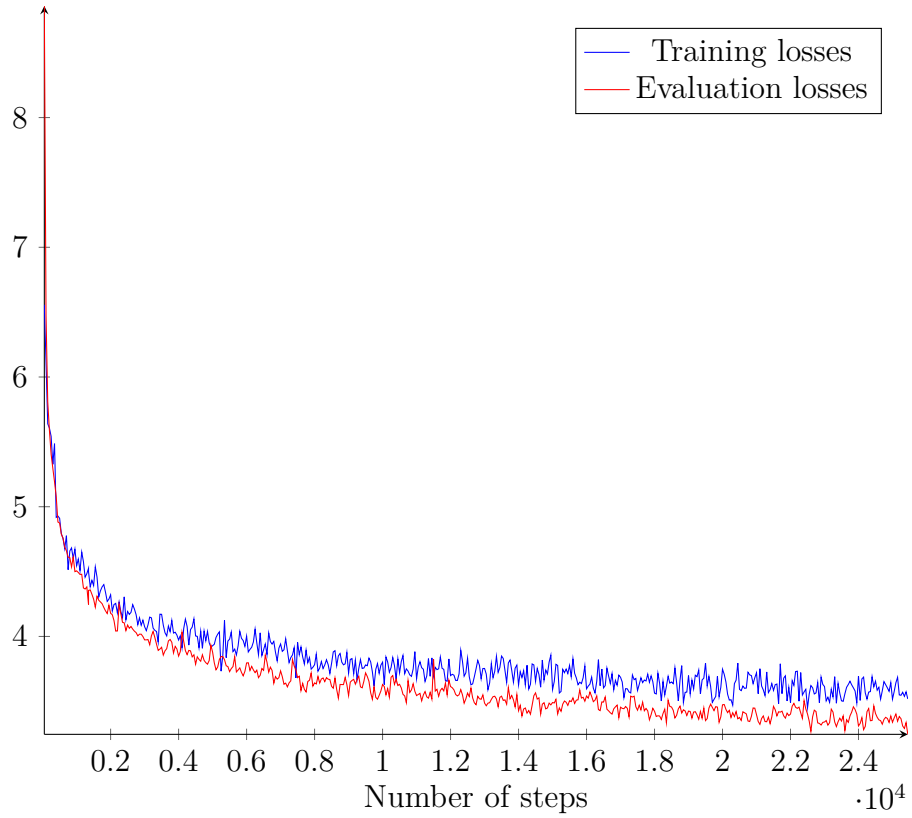

Figure 7.2: The figure shows our training losses in blue and evaluation losses in red during training for one of the models. As we can see both losses are still decreasing past step 14000, which is a sign that the network had not converged at the time the user evaluation were conducted.

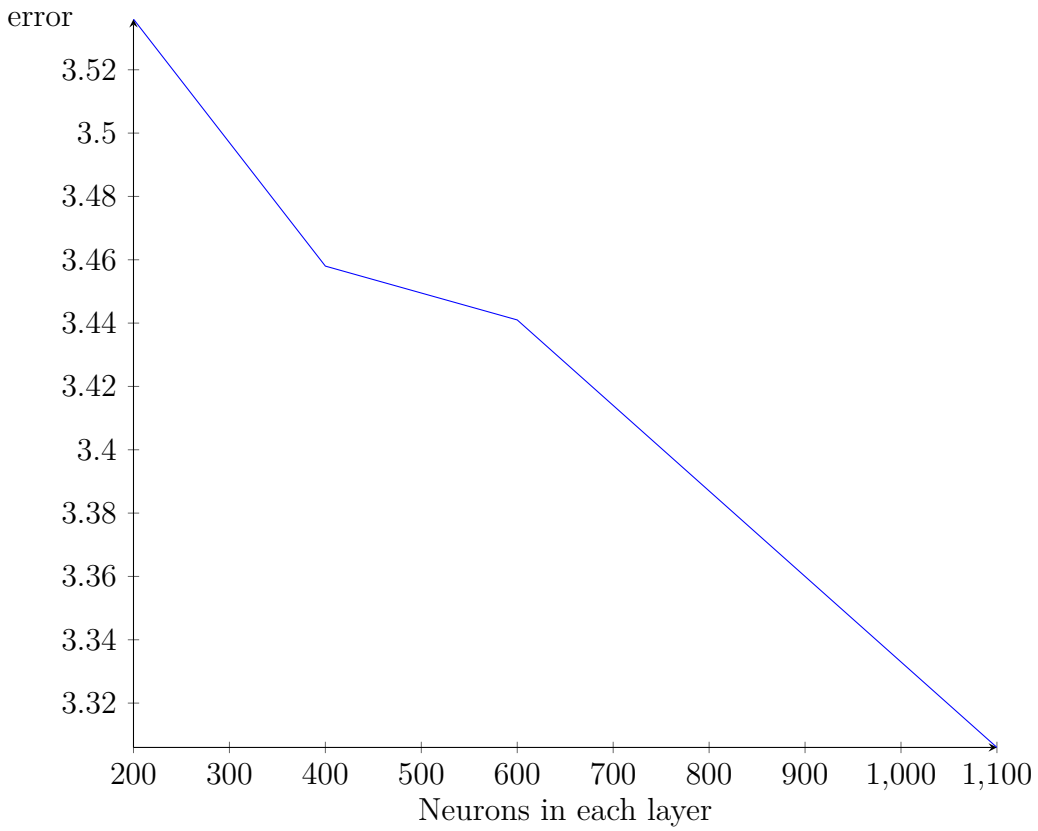

Figure 7.3: The validation error of four different network sizes after training of 24000 steps. Note that the lowest error is 3.306 and the highest is 3.536, so the difference in error between the smallest and the largest model is 0.23

layer [32]. Their findings seem to indicate that the bigger a network, the better it performs.

When Vinyals and Le performed a similar experiment as the one presented in this thesis, they used a two layered LSTM with 4096 cells in each layer [11]. In comparison, we also used a two layered network but with only 1700 cells in each layer. This is a significant difference so it is fair to believe that a larger network could perform better. It would be interesting to train the state-preserving model described in this thesis using more neurons, in the form of larger and perhaps more layers.

We did some additional testing on models of different sizes to see if it is reasonable to assume that a larger network size will increase the performance of our model. In figure 7.3 we demonstrate the evaluation errors of four models of different sizes that each has been trained for 24000 steps. Each of the models seemed to have converged, though the errors were still decreasing very slightly. What we can conclude is that larger network sizes seems to result in smaller validation errors, however the difference in error might be marginal. Additionally, since there seems to be no indication of the validation error to increases for larger sizes, we can conclude that there is room to test models with larger sizes without overfitting.

7.3.3 State-preserving model

Two models were trained and tested during the thesis work: one where hidden states were reset between every line of training and another where the hidden states was preserved between lines during a conversation. The idea behind the second model was to train the network on keeping a context during a conversation, so it could more easily remember earlier parts of it. However in the end, neither of the models were able to provide even a very basic conversation, and it therefore hard to know if this second model worked as we hoped.

7.4 Further work

Some new approaches are discussed here as further work. If one would train a model similar to the one described in this thesis, these suggestions could be taken into consideration.

7.4.1 Alignment and attention

The attention model that we use (see section 3.4.1) was initially used in a setting of translating sentences between different languages. It is not unreasonable to think that the same attention model would work for dialogue modeling. However, since we model conversations, a more appropriate attention mechanism would look not only on one input sentence, but perhaps on several input sentences. For example, the sum in equation 3.18 in section 3.4.1 could sum over all hidden encoder states from the whole conversation. The new sum would be as seen in equation 7.1, where τ is the number of words read by the encoder throughout the whole conversation (see equation 7.2) and α is some attention scaling factor which could be higher for newer sentences and lower for older sentences. However, this might become computationally expensive as the number of words in a conversation grows.

$$a'_t = \sum_{i=1}^{\tau} a_i^t h_i^L \alpha \tag{7.1}$$

According to Sutskever et al. [10], it is good to reverse input sentences when translating text using their sequence-to-sequence model. They showed this empirically with better translations, but their explanation was vague and more of a speculation. According to them, the average distance (measured in recurrent time steps) between words corresponding each other becomes lower when the input sentence is reversed. Unlike the data used for translation, our data set does not contain pairs where words at a specific position of either input or target necessarily corresponds to words at the same position of the other. We do not even know what "corresponds to" would mean in this context. In the worst case, since our data set is so noisy and unpredictable, reversing the input sentence could very well be detrimental to the result. However, there might exist some kind of general pattern of words corresponding to each other. Finding this pattern could potentially allow for a 'hack' similar to the reversing of input sentences, resulting in increased performance of the model. It would be interesting to see if just reversing the input would

yield any performance gains on a dialogue model. However, it is not trivial to what extent the input should be reversed. For example, either each line could be reversed individually, or whole conversations could be reversed.

7.4.2 Method of preserving the state

As described in section 4, the last state of the decoder is used as the initial state of the encoder. This has at least one problem. When the network switches from encoding mode to decoding mode, it is provided with a *GO* symbol. However, when a new sentence is fed to the network in the same conversation and the network switches back to encoding mode, there is no special symbol provided to the network that lets it know that this is happening. This could potentially degrade the quality of the output of the network. It would be interesting to test if feeding the network two different start symbols, one GO_{enc} for starting the encoding, and one GO_{dec} for starting the decoding.

Using the last state of the decoder is not necessarily the only way to preserve the state of the model. For example, the last state of the encoder could be used instead. The downside to this is that the network would not remember the response it had given for each input sentence. However, it might be easier to train the model when only preserving the encoder states, since it will not have to switch between encoder and decoder mode. The network will only remember the utterances of its conversation partner, but perhaps this is sufficient for creating a dialogue model. During a conversation, the state of the encoder at each input sentence will be affected by that it has read a stream of input sentences that start at the first sentence of the conversation. Thus, the encoder state at time t will now be h_τ. τ is defined in equation 7.2, where $X = (x_1, x_2, ...)$ is a conversation of sentences and $|x|$ is the length of the vector x.

$$\tau = \sum_{x_i}^{X} |x_i| \tag{7.2}$$

7.4.3 Method of partitioning data into conversations

In section 5.2 we talk about how we partition the OpenSubtitles data set into conversations. Recall that a new conversation is started after a set amount of time has passed since the last line. This time threshold was chosen by intuition. Thus, there might be a better threshold that divides the data set more accurately. However, no matter what threshold value we choose, there is always a possibility that one or more conversations get lumped together or cut of in the middle. An optimal data set would be annotated where each conversation begins and ends.

We believe that there might be better ways to split the data into conversations. One proposal that was out of scope for this thesis was to use unsupervised learning techniques, such as clustering algorithms to automatically partition the data set. Not only might the cut-offs be more accurate, but there might not even be a need to chose a global threshold value if the conversations are clustered together.

7.4.4 Out of vocabulary inference

During the interviews of the user testing, a frequent comment on several chatbots was that they did not remember the participant's name. This is expected for our model, since it does not process words that are not in the vocabulary and there is no guarantee that an arbitrary name will be in it. In feature-engineered chatbots, this can be easily solved by having a module that explicitly matches common phrases relating names, and simply memorizes the participants name. However, this is a problem that extends beyond remembering names. To have a general solution one must integrate it into the end to end trainable model.

Weston et al. [33] proposes a novel idea to model previously unseen words during inference and training of the model. The idea is based on the intuition that when humans find a word which they do not recognize, they will try to find out the meaning of it by looking at the context of where the word was used. In Weston's model, the context of a word is abstracted to be its neighboring words. One word to the left and one to the right of the unknown word is used as context, which means that the one-hot input vector will need to be three times the previous size. The three parts of the input represents the input word, the word to the left and the word to the right. As an example, with the vocabulary [how, is, $feeling$, ?] and the input sentence *"how is Boromir feeling?"*, the one-hot input vector of the third, unseen word *Boromir*, will be [0, 0, 0, 0 || 0, 1, 0, 0 || 0, 0, 1, 0] (here, || is used as the concatenation symbol). Training a neural network on this type of input should allow for it to learn to understand the semantics of sentences that contain an unknown word. However, our network would fail when queried about words which is out of vocabulary, such as asking *"what is my name?"* after having previously told it your name. Since it does not memorize specific words uttered to it, there is no possibility for the network to output a specific word which is not in its vocabulary. Weston uses a network architecture they call *memory networks* to explicitly store sentences, which makes it possible for the network to remember the actual words.

8

Conclusion

The final model presented in this thesis did not meet our initial expectations. We expected to produce at least a dialogue system capable of having very basic conversations, similar to the results achieved by Vinyals and Le. [11]. We proposed a few reasons as to why we did not get the expected result, but our belief is that the main issue is the small size of our model combined with a too short training time.

The chatbot did however show some results. We managed to train a model that always generates complete and correct sentences, though they do not make sense in the context of the conversation. The model also answers appropriately to common phrases such as greetings and valedictions, showing that the chatbot has at least some potential to learn to produce relevant responses.

Deciding on a method to evaluate the dialogue system was a significant part of the work in this thesis. Evaluating how well a dialogue system converses, as it turns out, is a hard task. The method we settled on using seemed appropriate for our specific setting. Since the systems are scored on many different areas, it is possible to find the strengths and weaknesses of the different systems. However, if one would need to compare a large number of dialogue systems, this method would get too unwieldy. Our belief is that more research needs to be made in the area of evaluating dialogue systems with focus on how natural the dialogue is.

Even though the final model did not meet our expectations, we believe dialogue systems trained with machine learning techniques has potential and that it is definitively worth investigating on how to improve them.

Bibliography

[1] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. `http://nlp.stanford.edu/projects/glove/`, 2014 (accessed May 26, 2016).

[2] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[3] Joseph Weizenbaum. Eliza; a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, January 1966.

[4] Rollo Carpenter. Cleverbot.com - A Clever Bot - Speak to an AI with some actual intelligence? `http://www.cleverbot.com/`, 2015 (accessed March 14, 2016).

[5] The A.L.I.C.E. AI Foundation. Free A.I.M.L. Alice set. `https://code.google.com/archive/p/aiml-en-us-foundation-alice/`, 2011 (accessed March 14, 2016).

[6] Creative Virtual. Smart help solutions - Improve Customer Service & Customer Engagement. `http://www.creativevirtual.com/`, 2016 (accessed May 25, 2016).

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[8] Byron Reeves and Clifford Nass. *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places*. Cambridge University Press, New York, NY, USA, 1996.

[9] Timothy Bickmore and Justine Cassell. How about this weather? Social Dialogue with Embodied Conversational Agents. In Kerstin Dautenhahn, editor, *Socially Intelligent Agents: The Human in the Loop*, number FS-00-04 in AAAI Technical Report, pages 4–8, Menlo Park, California, 2000. AAAI Press.

[10] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[11] Oriol Vinyals and Quoc V. Le. A neural conversational model. *CoRR*, abs/1506.05869, 2015.

[12] David Ameixa, Luisa Coheur, Pedro Fialho, and Paulo Quaresma. Luke, i am your father: Dealing with out-of-domain requests by using movies subtitles. In Timothy Bickmore, Stacy Marsella, and Candace Sidner, editors, *Intelligent Virtual Agents: 14th International Conference, IVA 2014, Boston, MA, USA,*

August 27-29, 2014. Proceedings, pages 13–21, Cham, 2014. Springer International Publishing.

[13] Marilyn A. Walker, Diane J. Litman, Candace A. Kamm, Ace A. Kamm, and Alicia Abella. Paradise: A framework for evaluating spoken dialogue agents. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 271–280, 1997.

[14] Ryuichiro Higashinaka, Masahiro Mizukami, Kotaro Funakoshi, Masahiro Araki, Hiroshi Tsukahara, and Yuka Kobayashi. Fatal or not? Finding errors that lead to dialogue breakdowns in chat-oriented dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2243–2248, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[15] Cameron Smith, Nigel Crook, Johan Boye, Daniel Charlton, Simon Dobnik, David Pizzi, Marc Cavazza, Stephen Pulman, Raul Santos De La Camara, and Markku Turunen. Interaction strategies for an affective conversational agent. In *Proceedings of the 10th International Conference on Intelligent Virtual Agents*, IVA'10, pages 301–314, Berlin, Heidelberg, 2010. Springer-Verlag.

[16] David Benyon, Björn Gambäck, Preben Hansen, Oli Mival, and Nick Webb. How was your day? Evaluating a conversational companion. In *IEEE Transactions on Affective Computing (Volume:4 , Issue: 3)*, pages 299–311, London, England, 2013. IEEE.

[17] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[18] Robert Stufflebeam. Neurons, synapses, action potentials, and neurotransmission. `http://www.mind.ilstu.edu/curriculum/neurons_intro/neurons_intro.php`, 2008 (accessed June 13, 2016).

[19] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[20] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[21] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, March 1994.

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[23] Rafal Józefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In Francis R. Bach and David M. Blei, editors, *ICML*, volume 37 of *JMLR Proceedings*, pages 2342–2350. JMLR.org, 2015.

[24] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[25] Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. Grammar as a foreign language. *CoRR*, abs/1412.7449, 2014.

[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[27] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.

[28] Kågebäck Mikael and Hans Salomonsson. Word sense disambiguation using a bidirectional lstm. *ArXiv e-prints*, 1606.03568, June 2016.

[29] Jörg Tiedemann. News from OPUS - A collection of multilingual parallel corpora with tools and interfaces. In N. Nicolov, K. Bontcheva, G. Angelova, and R. Mitkov, editors, *Recent Advances in Natural Language Processing*, volume V, pages 237–248. John Benjamins, Amsterdam/Philadelphia, Borovets, Bulgaria, 2009.

[30] OpenSubtitles. Subtitles - download movie and TV series subtitles from the biggest open subtitles database. `http://www.opensubtitles.org/`, 2016 (accessed May 26, 2016).

[31] David Benyon and OliMival. Landscaping personification technologies: from interactions to relationships. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 3657–3662, Florence, Italy, 2008. ACM.

[32] Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *CoRR*, abs/1602.02410, 2016.

[33] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *CoRR*, abs/1410.3916, 2014.

A

User testing questionnaire

The conversation between myself and the Chatbot felt natural

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

I thought the conversation was appropriate

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

I thought the Chatbot remembered earlier parts of the conversation

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

I liked the behaviour of the Chatbot

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

I felt I could correct the Chatbot when necessary

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

I found the Chatbot entertaining

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

I found the Chatbot engaging

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

I feel that the Chatbot is trustworthy

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

Over time I think I would build up a relationship with the Chatbot

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

I thought the Chatbot acted independently

- Strongly Disagree
- Disagree
- Undecided
- Agree

- Strongly Agree

The Chatbot's responses were always appropriate

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

The Chatbot showed empathy towards me

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

The Chatbot had an open and agreeable personality

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

The Chatbot demonstrated emotion at times

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

The Chatbot got to know me during the conversation

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

The Chatbot has a sensible attitude

- Strongly Disagree
- Disagree
- Undecided

- Agree
- Strongly Agree

The Chatbot is rather like me

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

The Chatbot surprised me at times

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

The Chatbot anticipated my needs

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

The conversation was coherent

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

The Chatbot was polite

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

The Chatbot was humorous

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

The Chatbot was friendly with me

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

The Chatbot is a good listener

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

I found the Chatbot boring

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree

I thought the Chatbot was responsive

- Strongly Disagree
- Disagree
- Undecided
- Agree
- Strongly Agree