NTNU
Norwegian University of
Science and Technology

Exploring Cells and Context Approaches for RNN Based Conversational Agents

Silje Christensen
Simen Johnsrud

Master of Science in Computer Science
Submission date: June 2017
Supervisor: Massimiliano Ruocco, IDI

Norwegian University of Science and Technology
Department of Computer Science

Task Description

The objective of the master's thesis is to explore conversational agents and study how we can use Recurrent Neural Networks (RNNs) to construct them. This thesis explores and compares the effectiveness of different RNN architectures, and investigates how we can handle context in conversational agents.

Abstract

Natural Language Processing is a challenging field within Artificial Intelligence, and building bots and conversational agents have been pursued by many researchers over the last decades. These agents should output reasonable responses, based on user inputs. In this thesis, we give an introduction on how to create conversational agents based on the current state-of-the-art, using Recurrent Neural Networks (RNN). We delve into different RNN architectures and compare the quality of the outputs from nine distinct agents. The baseline is an Encoder-Decoder model using Long Short-Term Memory (LSTM) cells, which is fed with question-response pairs. We compare it with models consisting of other RNN cells and explore different approaches that take the context of the entire conversation into account. To evaluate the models, we trained them on two different datasets. Five models were trained on the Ubuntu Dialogue Corpus (UDC), whereas four were trained on the OpenSubtitles dataset. The UDC is a closed domain dataset which is suitable when we aim for a beneficial conversational agent for a specific area. The OpenSubtitles dataset, on the other hand, is an open domain dataset and is used to capture how well the models handle chit-chatting (casual conversations and small talk). To feed the models with proper training data, we propose a procedure which preprocesses the data in four steps. One of the advantages of this preprocessing procedure is the removal of unknown tokens. This means that the training data only consists of words that exist in the conversational agents' vocabulary. The results indicate that the use of Grid LSTM cells improve the quality of the responses for the chit-chatting task and that the use of a context-based model generates responses which reflect the topic of the conversation.

Sammendrag

Naturlig språkprosessering (Natural Language Processing) er et utfordrende felt innen kunstig intelligens, ettersom det er vanskelig for en maskin å forstå og generere språk. Flere forskere har undersøkt hvordan man kan utvikle samtaleagenter, også kalt chatbotter, i løpet av de siste tiårene. Disse agentene skal generere en respons basert på et spørsmål fra en bruker. I denne masteroppgaven gir vi en introduksjon til hvordan man kan lage samtaleagenter ved å benytte oss av de nyeste arkitekturene for å konstruere Rekurrente Neurale Nettverk (RNN). Vi undersøker ulike RNN arkitekturer og sammenligner kvaliteten på svarene generert av ni ulike chatbotter. Vårt utgangspunkt er en Encoder-Decoder modell bestående av Long Short-Term Memory (LSTM) celler som blir matet med spørsmål-respons-par. Denne blir sammenlignet med modeller bestående av Gated Recurrent Unit (GRU) celler, Grid LSTM celler, og andre RNN konstruksjoner som har som formål å håndtere konteksten av hele samtalen. Vi har brukt to ulike datasett for å evaluere modellene. Fem ble trent på Ubuntu Dialogue Corpus (UDC), og fire ble trent på OpenSubtitles datasettet. UDC blir kategorisert som et lukket-domene-datasett (closed domain). For å lage en nyttig agent som skal kunne besvare spørsmål innenfor et spesifikt tema, er man nødt til å benytte seg av et slikt datasett. OpenSubtitles-datasettet er derimot et åpent-domene-datasett (open domain). Dette datasettet blir brukt for å teste chatbottenes evne til å håndtere hverdagslige samtaler og småsnakk. For å mate samtaleagentene med best mulig treningsdata, har vi definert en prosedyre som prosesserer datasettene i fire steg. En av fordelene ved denne prosesseringen er at vi får dekket all treningsdata med vokabularet til samtaleagentene. Resultatene indikerer at agenter basert på Grid LSTM celler genererer bedre responser enn de andre agentene når de blir testet på hverdagslige samtaleemner. Videre viser resultatene at en agent som er basert på en kontekst-modell klarer å generere responser som reflekterer temaet for samtalen.

Preface

This thesis was written spring 2017 for the Department of Computer Science and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU).

The subject for the thesis was defined in cooperation with our supervisor Massimiliano Ruocco. Several persons have contributed with inspiration and ideas for this project. First, we would like to express our gratitude to our supervisor Ruocco, and our co-supervisors Axel Tidemann and Cyril Banino-Rokkones for valuable input, assistance, and feedback throughout the project. Furthermore, we would like to thank Faez Shakil for helping us with the implementation of Bidirectional Grid LSTM.

[illegible] [illegible]

Silje Christensen and Simen Johnsrud
Trondheim, June 20, 2017

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

This chapter is meant as an introduction to the master's thesis. Section 1.1 states the fields this projected is situated in and the key driving forces that motivate the research. The research's goal and questions are defined in Section 1.2, whereas Section 1.3 further elaborates the research methods applied for the project. Then, a brief summary of the main contributions is provided in Section 1.4, and finally, we define the thesis' structure in Section 1.5.

1.1 Background and Motivation

This thesis studies how we can implement conversational agents using Recurrent Neural Networks (RNN), situated in the field of Artificial Intelligence (AI) and Natural Language Processing (NLP). A conversational agent, also referred to as a chatbot, is a computer system designed to communicate with humans. Based on the input from the user, it should generate a meaningful response. The agent should take care of the contextual aspects within a conversation, besides the language's grammar and semantics. The grammar includes the morphology, syntax, and spelling. Morphology is the internal structure of words, i.e. how words are formed, whereas the syntax focus on how words are related to each other. The semantics, however, tells the true meaning of a word or a sentence. The context captures the information received during a conversation and influences the way an expression is interpreted [Duranti and Goodwin, 1992]. Details from the past in the conversation is assumed to be known as the dialogue evolves.

What expectations we have to a chatbot, depends on its purpose. Which topics a conversational agent can converse within, is specified by the kind of dataset it is trained on. We differ between open domain and closed domain datasets. General knowledge and a rational behavior should be the expectation when interacting with an agent trained on an open domain dataset. It is should handle casual conversations and small talk, also referred to as chit-chatting. A task-oriented and useful conversational agent needs a more specific dataset and must be trained on a closed domain dataset. A closed domain chatbot will act in a special environment with clear boundaries on which topic it will manage to give reasonable responses to. In this case, the agent should behave as an expert in the field it is trained within. This kind of chatbot could automate several processes and be beneficial for industries that employ customer support. Unfortunately, today's technology has not been able to create conversational agents good enough to replace human beings in customer support.

AI is an exhilarating research field, and we find it interesting to study how a machine can learn to generate a grammatically correct response that answers a question. Thus, we want to explore the current state-of-the-art in the field of conversational agents and see whether changes in the presented models will affect the quality of the output from a chatbot.

1.2 Goals and Research Questions

The main goals of this research project are to compare the output from different RNN based conversational agents, and to incorporate context management to the models. To reach these objectives, we need to get in-depth knowledge of the current state-of-the-art in the field of conversational agent architectures. The thesis goal and its respective research questions are listed below.

Goal 1 *Explore and compare the effectiveness of different RNN architectures.*

Research question 1 *Will the use of different RNN cells in an Encoder-Decoder model have significantly effect on the quality of the outputs?*

Goal 2 *Incorporate context management in conversational agents.*

Research question 2 *Will a conversational agent that keeps track of previous questions and responses, catch the context of a conversation better?*

We will implement an RNN Encoder-Decoder model using LSTM cells as a baseline, and compare this model with the experimental architectures. As the automatic evaluation of conversational agents is still an open problem, we will evaluate them using perplexity in addition to conduct a human evaluation.

1.3 Research Method

The research process for our master's thesis is illustrated in Figure 1.1. To answer the research questions, we have followed an experimental research strategy, as de-

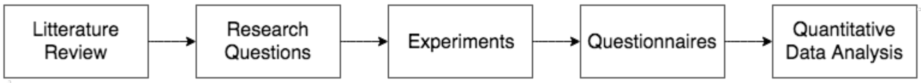


Figure 1.1: Overview of the research process

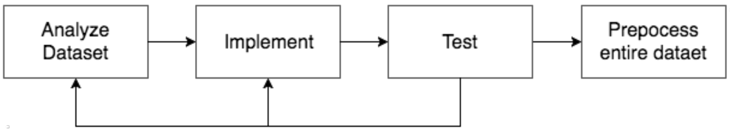


Figure 1.2: Experimental process - Preprocessing

scribed in Oates [2005]. For the first research question, we compare different RNN cells in a conversational agent. To answer the second research question, further changes are required. We change the preprocessing procedure, handle the internal state during training and modify the decoder. The effect of these changes is then compared to each other. In order to get data for analysis, we either choose questions from a part of the dataset that is not used for training or create fictive conversations to see how the different conversational agents reply. These results are applied in a questionnaire using Google Forms[1], where the evaluators will rate each answer with a score from 1 to 5. This survey gives us quantitative data which we will analyze to deduce a conclusion to the research questions. The results are affected by the dataset the model is trained on; thus, we increase the external validity by training some of the models with another dataset.

Every dataset needs adjustments to fit the proposed structure of the model's input. This preprocessing procedure is considered as a part of the experiment and is illustrated in Figure 1.2. We followed an iterative approach consisting of three phases. The first phase was to research and study the datasets in order to understand the changes required to improve the quality of the dataset. The second phase was to implement these changes, and the third to test it by running the new

[1]https://www.google.com/forms/about/

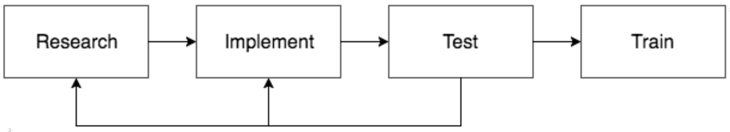


Figure 1.3: Experimental process - Model implementation

preprocessing procedure on a subset of the dataset. Based on the results, we would either go back and analyze the dataset further or continue working on the current task. Finally, we preprocess the entire dataset.

An illustration of the iterative approach used when implementing the conversational agents is depicted in Figure 1.3. Before we start the implementation, we need to research each desired model. If we can run the code and observe that the perplexity decreases, we start to train the model with the desired hyperparameters.

1.4 Contributions

The main contributions are:

1. A Stateful model, which is a result of our research on how adjustments to the Encoder-Decoder model can improve the model's ability to use information from previous turns. The conversations generated by this model and the results from the human evaluation show that it outperforms the LSTM baseline when it comes to the content.

2. A comparison of the Grid LSTM, LSTM, and GRU cells applied in the same architecture. The results indicate that the choice of RNN cells can affect the quality of the output and that the Grid LSTM cell increases the content quality of the responses in the chit-chatting task.

3. An approach for eliminating out-of-vocabulary (OOV) words from the dataset. The most common way to handle OOV words is to replace them with a unique unknown token, e.g. "UNK". Our approach suggests using a word representation model to represent all words with word embedding vectors. All OOV words will be replaced with the closest in-vocabulary word, by calculating the cosine similarity.

4. An architectural overview of the state-of-the-art models which can be applied when creating conversational agents.

We will elaborate the contributions of the research in Section 7.1.

1.5 Thesis Structure

The remainder of the thesis is structured as follows:

Chapter 2: Background Theory provides the reader with the knowledge required to understand the remaining chapters.

Chapter 3: Proposed Architectures presents six RNN architectures used to create the nine distinct conversational agents presented in this thesis. We explain how we implemented these architectures.
Chapter 4: Datasets describes the characteristics and shortcomings of both the Ubuntu Dialogue Corpus and OpenSubtitles dataset.
Chapter 5: Experimental Settings explains how we prepared, measured, planned and conducted the experiments. First, we define the preprocessing steps and the adjustments required for the different datasets. Then, we describe the evaluation metrics. Finally, we describe the experimental plan and setup and divide the experiments into three parts.
Chapter 6: Results and Discussion presents the results from each part of the experiments. We look at the perplexity results, the responses generated by the conversational agents, and the outcome of the human evaluation. Finally, we discuss the results.
Chapter 7: Conclusion and Future Work evaluates and concludes the presented work, by elaborating the contributions and answering the research questions. Further, it describes the challenges and provides a summary of further hypothetical research on conversational agents based on the work presented in previous chapters.

Chapter 2

Background Theory

In this chapter, we provide the reader with the necessary knowledge to understand the remaining content of the thesis. In Section 2.1 we explain why NLP is difficult, and further how a computer can use word embeddings to represent a word's semantic in order to understand and generate language. We explain Artificial Neural Networks (ANNs) and RNNs, and why the latter network is preferred for learning languages. The RNN Encoder-Decoder model can be used for creating a conversational agent, and this architecture is also presented in this section. Further, the basic method to train a simple feedforward network is then explained, followed by an explanation of the changes required to train an RNN. In Section 2.2 we study the current state-of-the-art, to get an overview of models used for conversational agents and the translation problem, which is a similar field. Then, in Section 2.3, we delve into the RNN architecture, to fully understand the existing approaches.

2.1 Background Theory

2.1.1 NLP and Word Embeddings

NLP is a challenging field within Computer Science and AI. It covers computer understanding, manipulation, and generation of human language. The computer beats the human brain in several areas, but when it comes to the processing of natural language, today's technology is not sufficient. Natural languages distinguish from formal languages, where the latter is a language consisting of a set of strings of symbols together with a set of rules that are specific to it, as in programming languages. Natural language, however, has evolved naturally in humans without intentional planning and is more complex. This makes languages inherently high dimensional, meaning that a word has several "characteristics". It can be positive

France 454	Jesus 1973	Xbox 6909	Reddish 11724	Scratched 29869	Megabits 87025
austria	god	amiga	greenish	nailed	octets
belgium	sati	playstation	bluish	smashed	mbs
germany	christ	msx	pinkish	punched	bits
italy	satan	ipod	purplish	popped	baud
greece	kali	sega	brownish	crimped	carats
sweden	indra	psNUMBER	greyish	scraped	kbits
norway	vishnu	hd	grayish	screwed	megahertz
europe	ananda	dreamcast	whitish	sectioned	megapixels
hungary	parvati	geforce	silvery	slashed	gbit/s
switzerland	grace	capcom	yellowish	ripped	amperes

Table 2.1: Word embeddings in the word lookup table of the language model neural network trained with a dictionary of size $100,000$ and with a 50-dimensional vector to represent each word. For each column the queried word is followed by its index in the dictionary (higher means more rare) and its 10 nearest neighbors (arbitrary using the Euclidean metric). [Collobert et al., 2011]

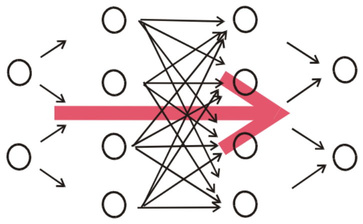
or negative, it might describe a noun or a verb; thus it cannot be described in few dimensions. Languages are also sparse, which means that a word does not necessarily have a high "score" on all these characteristics; thus most of the elements are zero. These are two of the reasons for why NLP is difficult. Another problem is that languages are changing every day, as new words or new terms occur. Further, it is hard for a computer to understand sarcasm, idioms, and ambiguity.

Although it is difficult for a computer to understand natural language, we can represent a word's semantic using word embeddings. This concept was originally introduced by Bengio et al. [2003]. A word embedding is the mapping from a word to an n-dimensional vector: $W \rightarrow \mathbb{R}^n$. This means that a word can be represented as a vector, and similar words are close together in the n-dimensional space. Table 2.1 illustrates how well word embeddings can represent the semantic of a word.

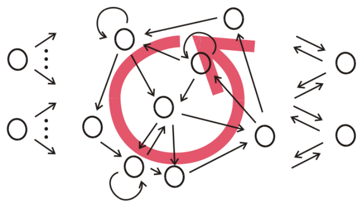
2.1.2 Artificial Neural Networks

ANNs are a class of machine learning algorithms that train on input data, to classify new unseen data in a similar domain. The research from the last decades has shown that these networks can solve a wide range of different tasks, from the translation problem to the image recognition task. The MNIST dataset[1] has become the classic example of how an ANN works, and how well it can perform certain tasks. This

[1]http://yann.lecun.com/exdb/mnist/



(a) Feedforward network. The processing stream of the network is following one direction, with neither cycles nor loops



(b) RNN network. The network consists of cycles, opening the possibility to revisit a neuron in the same layer

Figure 2.1: Feedforward network vs. Recurrent Neural Network. In each network, the units to the left shows the input units, while the rightmost is the output units. In between, there are layers of artificial neurons. The figure is from Jaeger [2002].

dataset consists of handwritten numbers from 0 to 9, and the ANN's function is to classify which number the current picture depicts. The network receives a picture (which in this case is of 32x32 pixels) as input and uses ten different output classes (one class for each number) to predict which number the input image illustrates. The output will be a vector telling the probability for the different classes, where the class with the highest probability will be the network's final answer. The task of recognizing numbers seems trivial to a human but is a lot more complicated for a computer. However, the best performing ANN for this task achieves a success rate of 99.79%, approximately 1.5% better than the average human [Wan et al., 2013].

ANNs are based on a large collection of artificial neurons, also referred to as units, and mimics the way our brain solves problems with large clusters of biological neurons. The signals and state of artificial neurons are real numbers, typically between 0 and 1. As the ANNs can be used to solve different tasks, there exist different kinds of architectures which have a more specified area of application. The feedforward network is the simplest kind of ANNs and is, in fact, the most popular subclass of ANN to handle image recognition tasks. For sequences of data, e.g. text, the Recurrent Neural Network (RNN) subclass is more favored. Their differences is illustrated in Figure 2.1.

The architecture is organized in layers, where each layer consists of the artificial units. The layers are divided into three categories: The input layer, the hidden layers, and the output layer. The input layer differs from the other layers, as they do not contain "fully artificial units", but rather the values in the data record,

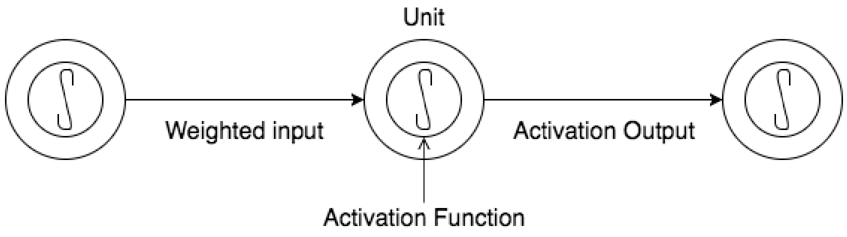


Figure 2.2: The unit is connected to other units through the weights. Each unit has an activation function which it uses to create output and hence input, to the connected unit.

which they pass on to the hidden layer(s). After traversing the units in the hidden layers, the signals reach the output layer which will classify the current input.

Synaptic links connect the units where the strength of the connection is defined by weights, working as a port deciding how much of the signal that should pass to the next unit. The weights' value will affect the final output. That is why these weights are adjusted during training in order to find an optimal solution for the network. The unit itself has an activation function, which assures that a small change in the weights will give a small change in the output of the unit. The input to the unit is a combination of the weight(s) and the input signal, i.e. activation output signal from the previous layer. This combination is referred to as the weighted input for a unit. Figure 2.2 illustrates the relation between the units, activation functions and the signals. We denote the activation function as $u(x)$ for input units, $f(x)$ for internal units, and $g(x)$ for output units [Jaeger, 2002]. Logistic, tanh, ReLU are examples of such functions [Goodfellow et al., 2016].

2.1.3 Recurrent Neural Networks

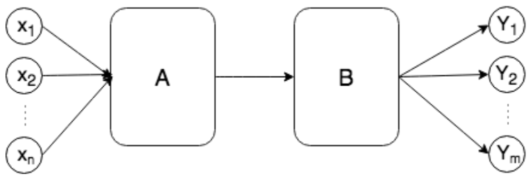


Figure 2.3: RNN with two hidden layers

RNN is a subclass of ANNs where the connections between the units form a directed cycle, as illustrated in Figure 2.1b. Figure 2.3 is an example of an RNN with two hidden layers, layer A and B. The layers are sometimes referred to as cells, and

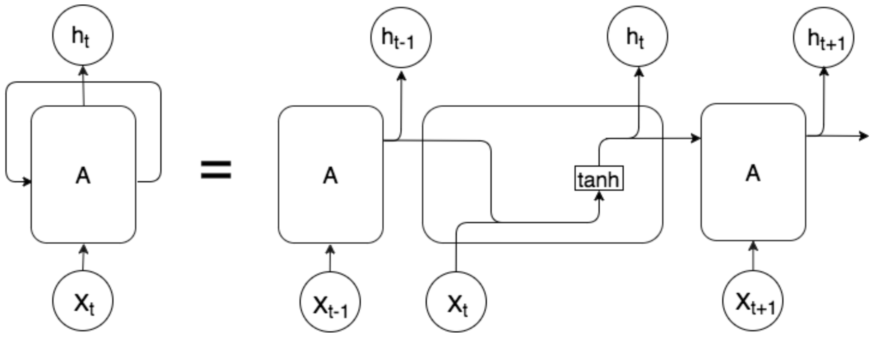


Figure 2.4: Unrolled RNN cell with tanh activation function

consists of “one unrolled unit”. If this network were used to create a conversational agent, it would be able to output m unique words, because it has m output nodes. This RNN unit has self-loops, as illustrated in Figure 2.4. The output from the unit at timestep $t-1$ is the input to the same unit at timestep t. Thus the new state is affected by the previous state when it handles the new input X_t. The activation function for a simple RNN could, for example, be tanh:

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1. \tag{2.1}$$

RNNs are designed to recognize patterns in sequential data. If a traditional feedforward network were to handle a sequence of words, it would have separate parameters for each input feature. This means that it would need to learn all of the rules of the language separately at each position in the sentence. In comparison, RNN shares the same weights across several time steps, making it possible for the model to remember what it learned from the first input when it handles the next within a sequence. In other words, RNN is a good choice when we aim to teach a computer to model and generate a language, as it can learn dependencies between all words in a sentence [Goodfellow et al., 2016].

In the case of conversational agents, the RNN is trained to predict the next symbol in a sequence. The output at each time step t is the conditional distribution

$$p(x_t|x_{t-1}, ..., x_1). \tag{2.2}$$

By combining these probabilities, we can compute the probability of the sequence x using

$$p(x) = \prod_{t=1}^{T} p(x_t|x_{t-1}, ..., x_1). \tag{2.3}$$

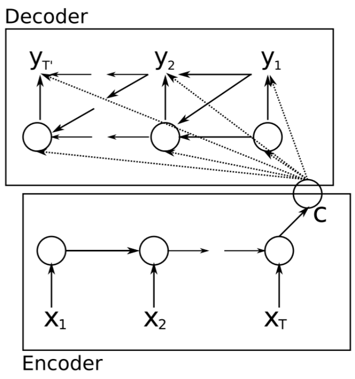


Figure 2.5: An illustration of the RNN Encoder-Decoder Architecture, from Cho et al. [2014].

The sequential information is preserved in the network's hidden state h. At each time step, t this state is updated by

$$h_t = f(Wx_t + Uh_{t-1}), \tag{2.4}$$

where x_t is the input at time step t, and W is the weight matrix, containing the weights that decide the strength of the signal from a unit. h_{t-1} is the hidden state of the previous time step, and U is the previous hidden state's transition matrix. The matrices work as “filters” and determine the importance of the current input and the former hidden state. The error these filters generate, found via backpropagation, will be used to adjust the matrices until the error is low. We simplify the equation for the hidden state by writing

$$h_t = f(x_t, h_{t-1}). \tag{2.5}$$

As in an ordinary neural network, the final layer is the output layer, where there is one node for each class. The hidden layer's activation function is $f(x)$ and the output of the network is $g(f(x))$. The function g should classify the input x_i to an output y_i. Thus, an RNN should map the input sequence $x_1, ..., x_n$ to an output sequence $y_1, ..., y_n$.

2.1.4 RNN Encoder-Decoder

Cho et al. [2014] proposed an Encoder-Decoder architecture, where both the encoder and the decoder consists of an RNN illustrated in Figure 2.5. The encoder will map an input sequence to a new compressed representation, while the decoder will generate a new sequence based on this compressed representation. The network learns to encode a variable-length sequence $x_1, ..., x_n$ to a fixed-length vector representation c, and then decode it back into a variable-length sequence $y_1, ...y_{n'}$. Note that n and n' may be different. The encoder sequentially reads each token x_t of the input until it reach an "end-of-sequence" symbol, and updates the hidden state h_t according to Equation 2.5. After encoding the source sentence, the hidden state of the RNN is now a summary c of the whole input sequence and a global representation of the sentence. The decoder updates its hidden state by taking the previously created word y_{t-1} and the context c into account:

$$h_t = f(h_{t-1}, y_{t-1}, c). \tag{2.6}$$

The decoder can then calculate the variable-length sequence by computing the distribution of the next token,

$$P(y_t|y_{t-1}, y_{t-2}, ..., y_1, c) = g(h_{t,y_{t-1}}, c). \tag{2.7}$$

2.1.5 Training ANNs

The main goal of training neural networks is to obtain the lowest possible error from the model, without overfitting the training data. Overfitting occurs when the model starts to overreact to minor fluctuations in the training data, which results in a poor predictive performance[2]. The error of a model is calculated with an objective function, also referred to as a loss function, which measures the error between the predicted output and the expected output. The expected outcome is, in supervised training, given by the dataset as it is structured with both training data and output labels. Without the labels, the model is obligated to train unsupervised. In this case, the task is to infer a function to describe hidden structure from "unlabeled" data and is more complicated than the supervised case. As our conversational agents are fed with question-response pairs, we will focus on supervised learning in this section.

Before we study how we can train an RNN, let us run through something less advanced: training a simple feedforward network with three layers, as depicted in Figure 2.6. The three layers are connected by two weight matrices that will be adjusted during training to minimize the error between the predicted output and the truth. Each weight is denoted with three indexes. w^l_{jn} represents the weight going to the jth unit in layer l, from the nth unit in layer $l-1$. The error is computed in the final layer (the output layer) by a loss function C. The backpropagation algorithm moves backward from the output error, through the weights of each hidden layer, and assigns each weight a responsibility for a portion of the error by calculating their partial derivatives. The training process will incrementally

[2]https://en.wikipedia.org/wiki/Overfitting

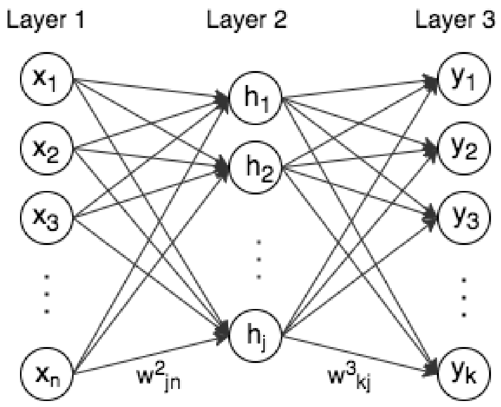


Figure 2.6: ANN with one hidden layer, and two weight matrices.

change the weights along the direction of the error gradient, i.e. the derivative. The weights are adjusted in the direction that lowers the error. The entire training process can be summarized in 5 stages:

Input
Every unit needs an activation function. The logistic function f can be used,

$$f(x) = \frac{1}{1 + e^{x}}, \tag{2.8}$$

which needs to be substituted and defined for a single unit as

$$y(u) = \frac{1}{1 + e^{w^T u}}. \tag{2.9}$$

The derivative will be used later in the backpropagation:

$$\frac{\partial y(u)}{\partial} = y(u)(1 - y(u)) = y(u)y(-u). \tag{2.10}$$

The goal of the first step is to compute the output activation for the first layer, denoted as a^1. This is just raw data fed into the input layer. However, the input data must have a certain structure, so that the model can handle it. In our example, we have n input units, which would match an input vector with n entries.

Feedforward
When the input layer has its corresponding activation, the values will be forwarded

to each succeeding layer:

$$\begin{aligned} z^2 &= w^2 a^1 \\ a^2 &= y(z^2) \\ z^3 &= w^3 a^2 \\ a^3 &= y(z^3). \end{aligned} \tag{2.11}$$

Output error
We can use the quadratic cost function that measures the loss between the true value t and the predicted output a as the loss function:

$$C = \frac{1}{2} \sum_k (t_k - a_k^L)^2. \tag{2.12}$$

The error in the output layer is then given by

$$E_k^3 = \frac{\partial C}{\partial a_k^3} y'(z_k^3) = a_k^3 - t_k. \tag{2.13}$$

Backpropagation
Now, we can backpropagate the previous result to find the error in the hidden layer

$$E_j^2 = \sum_k w_{kj}^3 E_k^3 y(z_j^2) y(-z_j^2). \tag{2.14}$$

Gradients
In the last step, we need to update the weight matrices such that the loss is minimized. This is done by an optimizer, e.g. Stochastic Gradient Descent (SGD). We need to find the gradient of the loss function, with respect to the weight vector w_j,

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} E_j^l. \tag{2.15}$$

Then we can run SGD to update the weights:

$$w^{new} = w^{old} - \frac{\eta}{m} \sum \frac{\partial C}{\partial w_{jk}^l}, \tag{2.16}$$

where $\eta > 0$ is the learning rate. The training pairs (x,y) are fed into the network until the weights converge to their optimal value. This is how we use SGD to learn to predict unseen data in a feedforward network.

Ruder [2016] studies different gradient descent optimization algorithms. Choosing a proper learning rate for SGD can be challenging, but in newer optimized gradient decent algorithms, such as Adagrad and Adam Optimizer, this is embedded in the algorithm. While SGD will use the same learning rate for all of the training parameters, Adagrad [Duchi et al., 2011] assigns a learning rate to all units, instead of having a single global one. As a result of Adagrad's behavior, frequent parameters will have a lower learning rate, i.e. get smaller updates, while it does larger updates on infrequent parameters, which makes Adagrad well suited for sparse data. The Adam optimizer [Kingma and Ba, 2014] supplies the Adagrad algorithm with an exponentially decaying average of past gradients, to gain faster convergence.

By expanding the simple example in Figure 2.6 with more hidden layers, we obtain a deep neural network with L layers. This, however, results in more weights matrices, which require a generalization of the training procedure, as listed below.

Input
Set the activation for the input layer; a^1.

Feedforward
For every layer 2,3, ..., L, compute the feedforward step

$$\begin{aligned} z^l &= w^l a^{l-1} \\ a^l &= y(z^l). \end{aligned} \tag{2.17}$$

Output error
In the output layer, we find the error

$$E_j^L = \frac{\partial C}{\partial a_j^L} y'(z_j^L). \tag{2.18}$$

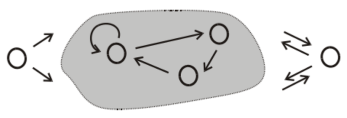
Backpropogation
For the layers $l = L-1, L-2, ...2$, we need to find the error in each layer.

$$E_j^l = \sum_k^K w_{kj}^{l+1} E_k^{l+1} y'(z_j^l). \tag{2.19}$$

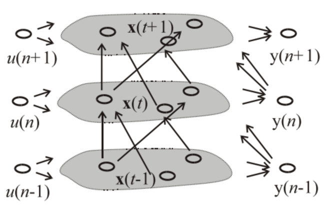
Gradients
The optimizer will finally adjust the weights by the gradients. The gradients to each weight matrix is found by

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} E_j^l. \tag{2.20}$$

The backpropagation algorithm assumes that there are no cycles in the network. Hence, it needs modification to be adopted by RNNs. The solution is Backpropagation Through Time (BPTT), an approach which unfolds the network in time.



(a) Original RNN



(b) Unfolded RNN

Figure 2.7: How BPTT unfolds an RNN to a feedforward network. Figure is from Jaeger's tutorial on RNNs Jaeger [2002].

An example of this approach is depicted in Figure 2.7. Figure 2.7a illustrates the original RNN, whereas Figure 2.7b is the transformed network, which functions as a feedforward network. Even though it has removed the cycles, the original connections between the units are obtained. [Jaeger, 2002]

The optimal way to train a neural network is to handle all the training data at the same time. If we could do this, it would be easier to point out the direction the weights should be moved, in order to reduce the overall error for the entire dataset. In that case, the model should converge faster, and the parameters would represent the optimal weights for the whole dataset. However, based on the computational power and memory required to handle all the training data simultaneously, this is impossible for all datasets with reasonable size. Instead, we are training in "batches". Each batch consists of n training pairs (a random subset of the entire dataset) which is trained at the same time. The smaller the batch, the more difficult is it to choose optimal gradients, hence the training process needs more time to find the optimal solution. The trade-off for the batch size is therefore between memory allocation and training time.

The training process of a batch, later referred to as a step in a training process, means that the network has generated an output vector (a response in a conversational agent) for all the training pairs in the batch, and then adjusted the weights based on the error through BPTT. When the network has processed all the training data, one epoch is finished, and a new one starts.

2.1.6 Tuning Hyperparameters

The goal of tuning hyperparameters is to choose values that will optimize a measure of the algorithm's performance. Due to the hardware limitations on the server, we also need to check how different hyperparameters, such as the number of units in each layer, the vocabulary size, and the bucket sizes, affect the memory while running the training process on a GPU. Table 2.2 shows the memory usage of different model parameters.

Model	U	L	Vocab	Optimizer	Bucket size	Mem. (MB)
Grid	512	2	100k	SGD	45-65-90-140	11892
Grid	512	2	100k	SGD	**10-16-22-30**	4269
Grid	512	2	**30k**	SGD	10-16-22-30	3565
Grid	**1024**	2	30k	**AdaGrad**	10-16-22-30	8373
LSTM	1024	2	30k	AdaGrad	**18-28-38-60**	8366
Stateful	1024	2	30k	AdaGrad	10-16-22-30	4286

Table 2.2: Memory allocation for different hyperparameters. The "Mem." column represents the memory usage in megabytes.

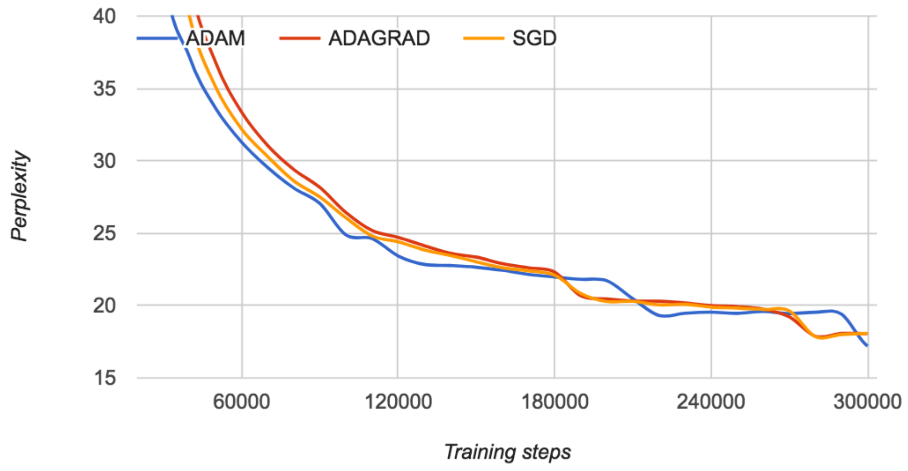


Figure 2.8: How the SGD, AdaGrad and Adam optimizer affect the perplexity

The experimentation with the SGD, AdaGrad and Adam optimizer does not confirm the theory presented in Section 2.1.5, where it is natural to assume that AdaGrad and Adam should be superior to the SGD. Figure 2.8 illustrates the perplexity results from the same model using different optimizers. The perplexity is a measurement of how well the model performs and is further explained in Section 5.2.1. We decided to use AdaGrad as our optimizer, as it is supposed to be better than SGD, and because the Adam optimizer required more time to train, due to its complexity.

2.2 State-of-the-art

In this section, we study the current state-of-the-art. There exist numerous of different papers concerning how to design conversational agents. The results have improved over time, but they also confirm the difficulties around NLP and the complexity of teaching a machine to hold a conversation like a human. In this section, we first look at newer word embeddings methods. Then, we study translation models, which are based on the RNN Encoder-Decoder model. Further, we take a closer look at recently proposed papers about conversational agents.

Mikolov et al. [2013a] presented the Skip-Gram model, which later was optimized in Mikolov et al. [2013b]. The training objective in the Skip-Gram model is to learn word vector representations that are good at predicting the nearby words. By looking at a given word, the algorithm will predict the surrounding words. This achieved significantly better results on syntactic and especially semantic accuracy. The other architecture presented in the paper, the Continuous-Bag-of-Words (CBOW) model, does the inverse, i.e. it will predict a target word based on the surrounding ones. CBOW achieves a slightly better score on the syntax, but the Skip-Gram model is superior when it comes to semantic accuracy. The improvements in Mikolov et al. [2013b] included computational speedups which are applicable to both the architectures. An example is to subsample frequent words, as these words do not contain as valuable information about the context in contrast to infrequent words. In addition to the computational time reduction, this also resulted in better word embeddings of rare words. Further, they presented a simple method to identify phrases in the text, and show that it is possible to learn good vector representations for millions of phrases as well. A phrase is a collection of words that have a different semantic meaning than the words have independently. Their example is "New York Times", consisting of three words, but referring to a particular newspaper. With these improvements, the models can train and learn on larger corpora than before.

By employing the information in the subwords of a word, i.e. the internal structure, the results have improved even more. Bojanowski et al. [2016] extends the model in Mikolov et al. [2013b] with a new scoring function that takes care of the internal structure of the words. They separate each word into smaller parts of 3-6 characters. These parts obtain their own word embedding, which is combined to form a word embedding for the entire word. Their model utilizes this to outperform the baselines, the skip-gram and CBOW models from the word2vec[3] library. Bojanowski et al. [2016] obtained great results on a 50M token dataset using 100-dimensional word embedding vectors, which has then become the default dimension in Facebook's fastText[4] library. FastText regroups this model with the

[3]https://code.google.com/archive/p/word2vec/
[4]https://research.fb.com/projects/fasttext/

efficient text classification presented in Joulin et al. [2016], resulting in a fast trainable word representation model for both small, rare and large datasets. The model will train on a corpus, and learn to output a word embedding for a new, possibly unseen, word. Unlike other lexical databases such as the WordNet [Miller, 1995], fastText gives the opportunity to train on any dataset, in order to retrieve word embeddings that fit the domain of the dataset.

Being able to represent a word's semantic is an essential building block in the process of creating a conversational agent. However, before we delve into this task, we study the translation problem, as it turns out that the same architecture can be applied to both of the problems. The difference between these two systems when using the same architecture lies in the training data. Instead of being fed with an English-Chinese pair, the conversational agent is fed with a question-response pair.

Sutskever et al. [2014] presented an RNN Encoder-Decoder related to Cho et al. [2014], for machine translation purposes. They used LSTM cells (Section 2.3.1) with a limited vocabulary which score better than a standard statistical machine translation (SMT) system with an infinite vocabulary. Their main contribution is the reversing of the input sequence, which improved the results significantly.

Bahdanau et al. [2014] discussed the existing architecture's weakness of handling long sentences. The challenge for the models was to convert the entire source sentence into a fixed-length vector, while still focusing on the important words in the sentence. Their solution was to add an attention mechanism to the Encoder-Decoder approach, which made it possible for the decoder to decide which parts of the source sentence to pay attention to. In order to achieve this, each word in the input should keep information about the entire input sequence, yet mainly focus on the surrounding words. The encoder calculates the attention vector, which is further used in the decoder. As the goal was to capture both the successive and preceding words, they changed the conventional encoder with a bidirectional RNN (BiRNN). A BiRNN consists of one forward and one backward RNN and was first described in Schuster and Paliwal [1997]. The forward RNN works as before, reading the input from the start to the end, while the backward does it in reverse, starting from the last word. Both of the RNNs will compute a hidden state each, which is concatenated to include all information acquired from traversing the sequence forward and backward. The concatenated state contains an "annotation" for each word, holding the summaries of both the preceding and following words. The annotations are used to create a context vector c_i, using Formula 2.21.

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \tag{2.21}$$

The weight α_{ij} of each annotation h_j is computed by using an alignment model, which scores how well the inputs and outputs match at certain positions. We have an attention mechanism because the α_{ij} signal reflects the importance of the annotation with respect to the previous hidden state in deciding the next state and generating the output. The results of their architecture show that the synergy between the BiRNN and the attention mechanism outperforms the standard Encoder-Decoder model, and increase the robustness to longer source sentences significantly.

Kalchbrenner et al. [2015] introduced Grid Long Short-Term Memory (Grid LSTM) as a new architectural combination of LSTM cells. This design arranges LSTM cells in N dimensions, which can be applied to several data structures including sequences. They did several experiments with their proposed model and observed the advantages of their architecture compared to the regular LSTM network. One of the experiments concerned the translation problem, and a two 3-dimensional Grid LSTM yielded good results. Unlike the simple LSTM architecture, the proposed Grid LSTM network will repeatedly scan the source sentence on each generated word. Another feature is that the source words and target words are projected on different sides of the Grid LSTM, ensuring that the vectors will interact closely without being conflated. We will describe the Grid LSTM in detail in Section 2.3.3.

Note that compared to the challenge of implementing a chatbot, the translation problem is significantly easier to solve. First, the sentences do not keep a context between each other, as they do in a conversation. Second, it is easier to work with as it has well-defined evaluation methods, e.g. BLEU [Papineni et al., 2002]. Automatic evaluation for a conversational agent is still an open problem. The research on conversational agents has resulted in two subcategories that distinguish between *retrieval* and *generative* based chatbots. The former architecture uses a repository of predefined responses, while the latter will generate new sentences it may not have encountered before. There are pros and cons to both approaches. Shang et al. [2015] and Vinyals and Le [2015] explores the generative, while Lowe et al. [2015] apply the retrieval approach.

Shang et al. [2015] propose a Neural Responding Machine (NRM) and employs the Encoder-Decoder framework to address the response generation problem, using GRU-cells. Unlike the previously mentioned research papers, they are looking at the Short-Text Conversation (STC) problem instead of translation. An STC is defined as a two-turn conversation, meaning that there are only two statements, one question and one response. The NRM is trained on a dataset collected from a microblogging service and is fed with post-response pairs. The decoder is a standard RNN model, where the input may differ slightly depending on which encoder it applies. They present three types of encoding schemes, a global scheme, a local scheme and a hybrid scheme which combines the two. The global scheme uses the same architecture as in Cho et al. [2014] and Sutskever et al. [2014], where the final hidden state in the encoder is the context, and then the input to the decoder. The local scheme is based on the approach described in Bahdanau et al. [2014], where the decoder can choose which parts of the input sequence to focus on. Both schemes have their pros and cons, and by concatenating the hidden state from the local and global decoder, they create a new approach, a hybrid scheme, which yields better results. Five persons conducted the human evaluation and assigned a score from 0 to 2 (unsuitable, neutral, suitable) to several responses given a post. The evaluators were asked to have grammar, fluency, logic consistency, scenario dependence, and generality in mind when evaluating the responses. They compared the NRM models with a Retrieval-based [Ji et al., 2014] and a statistical machine translation (SMT) based method [Ritter et al., 2011]. The SMT model performed significantly worse than the other candidates, while the retrieval based beat the NRM-global model. The hybrid scheme received the best results.

Vinyals and Le [2015] proposed "A Neural Conversation Model", using the architecture described in Sutskever et al. [2014]. They test the model using two datasets. The first dataset is a closed domain IT helpdesk troubleshooting dataset, extracted from chat logs. The second is an open domain movie transcript dataset. They evaluate the model using perplexity, which measures how well the model predicts a sample, in addition to a human evaluation. The output from the model shows that it sometimes manage to produce natural conversations. However, their model does not take the context of a conversation into account, which may result in contradicting replies within the dialogue.

Despite the obvious advantage of the generative model, there are several reasons for why the proposed generative models are not good enough for conversational agents. Even though it has decent results for short text conversations, it may struggle with longer sentences and conversations. Further, the grammar and sentence structure may be wrong. The latter problem is solved with the retrieval based model, as the sentences are picked from a fixed set, without grammatical errors. Several papers have looked towards this subcategory of conversational agents. Lowe et al. [2015] consider three different approaches to create a retrieval based model, a term frequency-inverse document frequency (TF-IDF) approach, an RNN approach, and finally they study an LSTM approach. Their architectures are built upon a Weakly Supervised Embedding Model from Bordes et al. [2014], initially used in the question answering task as described in Fader et al. [2013]. Bordes et al. [2014] transformed the questions and answers to vectors, and their objective is to make the question and the best-suited answer to be close in the n-dimensional space. The RNN model used in Lowe et al. [2015] consists of two RNNs. The first RNN finds the embedded vector for the question, where the last hidden states are used as a bias in the second RNN to find the embedded vector for the response using beam search[5]. The significant difference between Lowe et al. [2015] and the

[5]An algorithm that explores a subset of the model's possible outputs. It will traverse the n best paths until they end. (https://en.wikipedia.org/wiki/Beam_search)

previously mentioned papers is that they are concerned with the classification of responses, instead of generation. In addition to the RNN model, they considered the same architecture but changed the hidden units to LSTM units in order to model longer-term dependencies. Their results show that the LSTM model is significantly better than pure RNN and TF-IDF evaluating with the Recall@k metric. This evaluation method asks the model to output the k best responses, and it is correct if the correct answer is among these options.

2.3 RNN Architectures

We described the principles of a simple RNN in Section 2.1.3. This section will take a closer look at this network, and study how the RNN architectures have evolved. In the following subsections, we will describe the most common cells, i.e. the layer consisting of an artificial neuron with self-loops, applicable in the Encoder-Decoder architecture, and further look at extensions of these cells. The theory presented in this section, and the figures, are inspired by Colah's blog posts[6].

2.3.1 LSTM

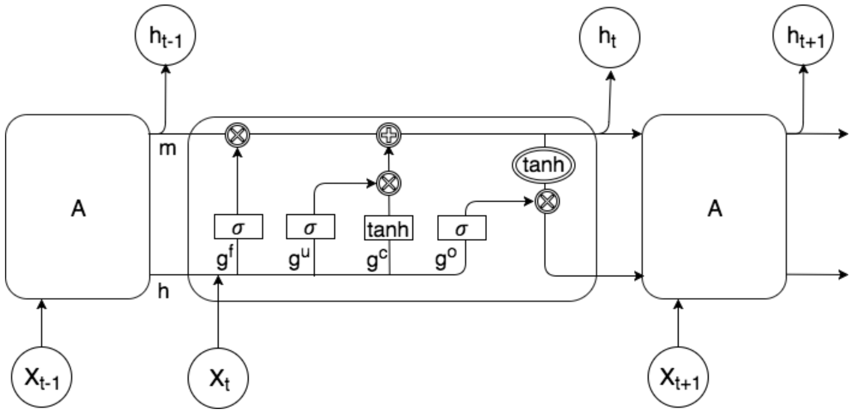


Figure 2.9: Unrolled LSTM cell.

Essential to the success of using RNNs is the use of Long Short-Term Memory (LSTM) cells. LSTM is an RNN architecture first proposed by Hochreiter and Schmidhuber [1997] and has been increasingly popular after its origin. RNNs using

[6]http://colah.github.io/

LSTM cells can learn long term dependencies to a greater extent than RNNs using standard cells. An unrolled LSTM cell is illustrated in Figure 2.9.

The cell comprises a memory state and a hidden state. The first state consists of all the information the network has at the current time step, whereas the hidden state is the networks internal memory. These states will be determined by the past inputs $(x_1, ..., x_{i-1})$. The computation at each step is defined in the equations in 2.22.

$$\begin{aligned}
g^u &= \sigma(W^u H) \\
g^f &= \sigma(W^f H) \\
g^o &= \sigma(W^o H) \\
g^c &= tahn(W^c H) \\
m' &= g^f \odot m + g^u \odot g^c \\
h' &= tanh(g^o \odot m')
\end{aligned} \tag{2.22}$$

The output of the cell is dependent on four mechanisms in which the cell is built upon. First, we have the forget gate mechanism, g^f, which decides how much of the old history in the state that should be carried on by deleting parts of the previous memory vector m_{i-1}. Further, the two mechanisms g^u and g^c will together determine the new memory m_i. g^u decides which values we will update, whereas g^c creates a vector of new candidate values that could be added to the state. In the next step, the LSTM cell combines these two to create an update to the memory state. The output gate mechanism, g^o, will then determine what is read from the new memory m_i onto the hidden vector h_i.

g^f, g^u and g^o consists of a sigmoid activation function, Equation 2.23, which outputs values between 0 and 1. If the output of the activation function is 1, it tells the gate to include all of the hidden information. If it outputs 0, it will get rid of this information. Since the gates can prevent the rest of the network from modifying the contents of the memory cells for multiple time steps, LSTM networks preserve signals and propagate errors for much longer than ordinary recurrent neural networks.

$$S(t) = \frac{1}{1+e^{-t}} \tag{2.23}$$

2.3.2 GRU

The recent years, Gated Recurrent Unit (GRU) [Cho et al., 2014], has been a popular substitute to LSTM. The GRU cell controls the flow of information in a similar manner as the LSTM cell. However, the cell consists of only two gates, the *reset* and the *update* gate, while the hidden state and the memory state is merged to one. By reducing the number of gates, the model consists of less complex cells, and hence speeds up the training process, as well as it is easier to implement. The reduction in complexity has made it to a popular alternative to the well-documented LSTM cells. Figure 2.10 illustrates the architecture of the cell.

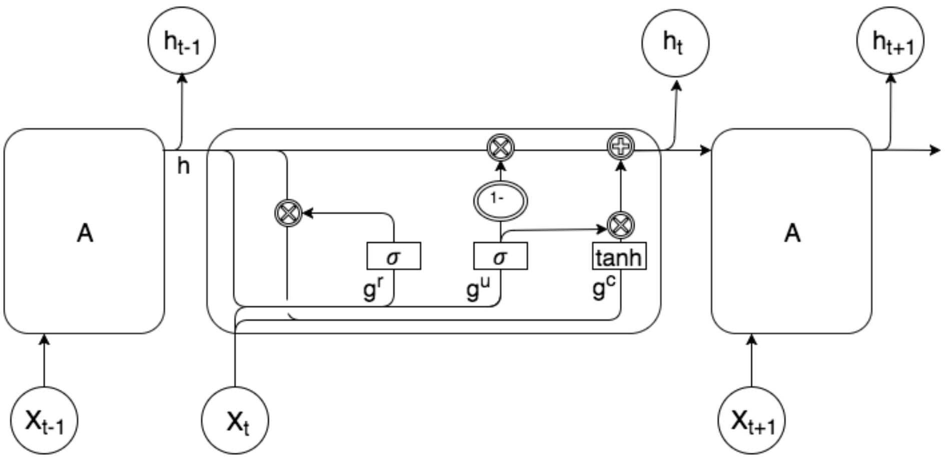


Figure 2.10: Unrolled GRU cell.

2.3.3 Grid LSTM

Kalchbrenner et al. [2015] proposed the N-dimensional Grid LSTM (N-LSTM) architecture. They motivate their work by the fact that other families of deep neural networks also suffer from the same problems as RNNs applied to long sequences. Unlike the LSTM architecture, other networks cannot dynamically select or ignore its inputs. Thus they want to generalize the advantages of LSTM to other deep computations. Further, they present a Grid LSTM architecture that yields good results on the translation problem.

The main difference between the standard LSTM cell and the Grid LSTM cell is the possibility for communication between layers. A feed forward network is similar to the 1-LSTM, while 2-LSTM is comparable to a stacked LSTM with cells along the depth dimension. The use of LSTM cells in an additional dimension is also the difference between Multi-Dimensional LSTMs [Graves et al., 2007] and Grid LSTMs with 3 or more dimensions.

To understand how the Grid LSTM architecture works, we need to look at the elementary units, which they refer to as blocks. A 2-LSTM block is illustrated in Figure 2.11. An unrolled block forms a 2-dimensional grid, consisting of two LSTM cells in each block. An N-Dimensional block will first concatenate the input hidden vectors from N dimensions as $H = [h_1, ..., h_N]$. Further, it will compute N LSTM transformations for each dimension, using the formulas in Equation 2.24. This means that both the new hidden and memory vector for the different dimensions vectors are distinct, yet they are influenced by a shared hidden vector H, as illus-

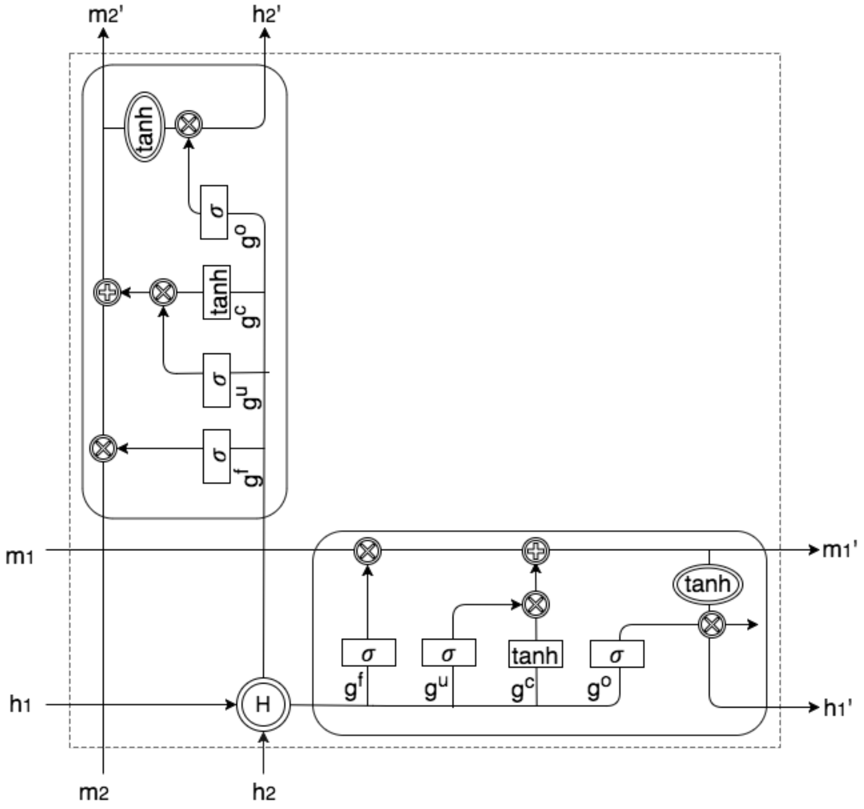


Figure 2.11: 2-Dimensional Grid LSTM block, consisting of two LSTM cells in different directions. When it is unrolled it will form a 2 dimensional grid.

trated in Figure 2.11. The blocks, and the grid that consists of blocks, will have N sides with incoming hidden and memory vectors, and N sides with outgoing vectors. This mechanism inside the blocks ensures that the hidden and memory vectors from the different sides will interact closely without being merged. The input is fed into the network on one of the sides of the grid as a pair of hidden and memory vectors, while the target is aligned along one of the other sides.

$$\begin{aligned}(h'_1, m'_1) &= LSTM(H, m_1, W_1)\\ (h'_2, m'_2) &= LSTM(H, m_2, W_2)\\ &...\\ (h'_N, m'_N) &= LSTM(H, m_N, W_N)\end{aligned} \tag{2.24}$$

If a network has a few blocks along a given dimension in the grid, it can be useful to just have regular connections along that dimension without the use of LSTM cells. Kalchbrenner et al. [2015] refer to this as a non-LSTM dimension. Another feature they present is the sharing of weight matrices along specified dimensions. If the weights are shared along all dimensions including the depth, it is defined as a Tied N-LSTM model.

Chapter 3

Proposed Architectures

This master's thesis introduces nine distinct conversational agents. However, some of them are based on the same model, as illustrated in Figure 3.1. This chapter is dedicated to present the implementation of six different RNN based conversational agents. First, the baseline described in Section 3.1 works as a basis for comparison, in addition to be the fundamental building block for the proposed models. Second, we explain the minor change that is required to create a GRU model. Further, the implementation of the Grid LSTM model is described in Section 3.3. Section 3.4 describes the Stateful model, whereas Section 3.5 describes the Stateful-Decoder model. Finally, Section 3.6 introduces the sixth model, the Context-Prepro model. Code snippets are presented to connect the implementation with previously described theory, to make it easier for the reader to see the differences between the proposed models, and to understand how different aspects of the Encoder-Decoder model works.

3.1 LSTM Baseline

The LSTM baseline is built to mimic the Neural Conversational Agent described in Vinyals and Le [2015]. This architecture adopts the Encoder-Decoder model from Sutskever et al. [2014], using LSTM cells. TensorFlow has proposed an interface for making this model and named it the Sequence-to-Sequence model[1]. This architecture is illustrated in Figure 3.2. The default model consists of LSTM cells that process one word at a time and compute the probabilities of the possible continuations of the sentence.

For this experiment, we adopt TensorFlow's Translation model[2], which provides

[1]https://www.tensorflow.org/tutorials/seq2seq

[2]https://github.com/tensorflow/models/blob/master/tutorials/rnn/translate/translate.py

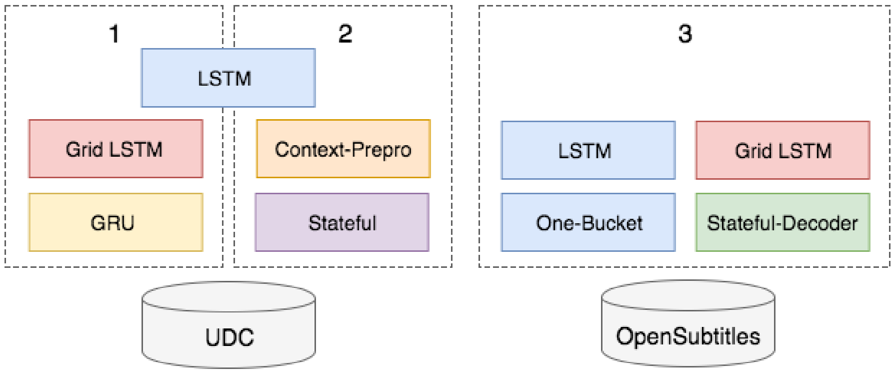


Figure 3.1: An overview of the nine conversational agents developed during this research project. Identical color implies that they have the same architecture, and the number indicates which experiment(s) they are used for. We have trained five chatbots on the UDC, and four on the OpenSubtitles dataset. The LSTM baseline, Grid LSTM, GRU, Context-Prepro, Stateful, and Stateful-Decoder are the six underlying conversational agent models and are described in Chapter 3. Section 5.3.3 explains the One-Bucket model.

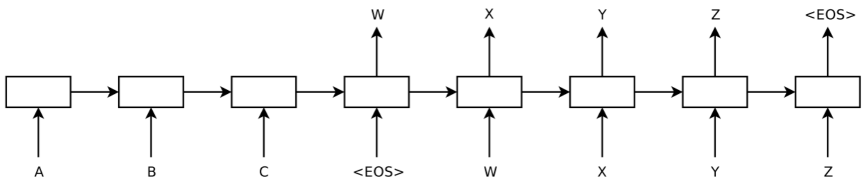


Figure 3.2: Sequence-to-Sequence architecture. Each box in the picture above represents a cell of the RNN. Image from Sutskever et al. [2014].

us with code for reading data, creating the model, training, and decoding for the translation task. These are the four core operations that every agent needs, and becomes the architectural interface for the models. The task of creating conversational agents is similar, but not identical, to the translation problem. Hence, the model requires changes to fulfill this functionality. The model uses the "embedding_attention_seq2seq" model, which is just one of many possible Sequence-to-Sequence architectures from the TensorFlow library. This architecture applies both the attention mechanism from Bahdanau et al. [2014], and the word embedding as described in Mikolov et al. [2013b], as presented in Section 2.2. It also reverses the input sentences as described in Sutskever et al. [2014].

The way our baseline prepare and feed the data is completely different from the proposed approach in the translation model. For this reason, Section 5.1 is dedicated to describe this. On the other hand, the creation of the model, the training process, and the decoding step are comparable to the methods in the translation model. Hence, we take a closer look at these functions in this section.

Create Model

The create_model function (Listing 3.1) is used to initialize or restore a model, and to prepare it for either training or interaction with a user. Further, the loss function and the BPTT is constructed and made ready for the training process. These operations, however, are not essential when interacting and evaluating the model. Hence, the create_model function has a parameter (forward_pass_only) which determines whether to execute the BPTT or not.

```
def create_model(forward_pass_only, ...):
    model = seq2seq_model.Seq2SeqModel(feed_forward, ...)
    return model
```

Listing 3.1: Create_model function

The model is built with the given number of layers, units, and the chosen RNN based cell. Listing 3.2 shows how the baseline is initialized with LSTM cells wrapped in multiple layers.

```
single_cell = tf.nn.rnn_cell.BasicLSTMCell(num_units)
cell = single_cell
if num_layers > 1:
  cell = tf.nn.rnn_cell.MultiRNNCell([single_cell] * num_layers)
```

Listing 3.2: Create LSTM model

The model requires fixed-length vectors of length n as input. This indicates that every question and response in the training data that has length i where $i < n$ needs to be zero-padded. If a sentence is short the input vector will mainly consist of padding tokens, which is time inefficient. To reduce some of this unnecessary padding, TensorFlow introduced "buckets" to their model. The purpose of the buckets is to allow multiple fixed-size vectors for the input and the output, so that shorter sentences can avoid many padding tokens. To further increase the efficiency, the model applies the sampled softmax, to avoid that training and decoding complexity increase proportionally to the vocabulary size.

Train

When the model is ready for training, it is fed with question-response pairs. The code in Listing 3.3 illustrates a simplified version of the training process. The model is handed an amount of training data, which it used to form a batch, and performs a training step. The training step is then calculating the loss and performing the BPTT arranged by the model creation.

```
    while max_train_steps >= current_step:
        encoder_inputs, decoder_inputs, target_weights = model.
    get_batch(train_set, bucket_id)
        _, step_loss, _ = model.step(encoder_inputs, decoder_inputs,
    target_weights, ...)
```

Listing 3.3: Training procedure

The feeding of training data to the model is done in batches. Our approach differs from the one used in TensorFlow's translation model. TensorFlow's approach loads the entire training file into memory and randomly picks samples to construct a batch. For massive datasets, this will require a lot of memory. Instead, our model continuously streams training data from a stored file meaning that the only training data that is loaded into memory, is the pairs that the model is currently training on. This comes with two advantages: First, the reduced memory could be used to extend the model further or increase the batch size. Second, this approach guarantees that all of the training data is used equally many times during training. When the file reader reaches the end of the file, i.e. one epoch is over, the training file is shuffled to assure that the batches for the next epoch are different from the previous.

Decode

Interaction with the chatbot is done by restoring a checkpoint from a trained model. As the model is not supposed to adjust weights when it decodes, it does a single step without the backward pass. The outcome of this step will then consist of the most predicted sequence, rather than the loss. A user will only type one input at the time. Therefore, the batch size must be adjusted to consist of a single sample. Listing 3.4 describes the decoding process.

```
    model = create_model(forward_pass_only=true, ...)
    model.batch_size = 1
    while interacting:
        question = preprocess(user_input, ...)
        encoder_inputs, decoder_inputs, target_weights = model.
    get_batch(question, ...)
        _, _, output = model.step(encoder_inputs, decoder_inputs,
    target_weights, forward_pass_only=true, ...)
```

Listing 3.4: Decoding procedure

The decoder applied in our baseline practice a greedy approach, which means that the most predicted word is followed by the next most predicted word etc. We discuss an optional approach for the decoder in Section 7.4.

The question a user inputs to a conversational agent, should ideally have no grammatical errors. This, however, is hard to ensure. Therefore, the input to the conversational agent will walk through some of the preprocessing steps, which will be described in Section 5.1. Recall that the chatbot requires input and output as vectors with fixed length. As a result, the chatbot will continue to generate words until the current bucket size is reached. The output may of this reason, consist of several sentences. We observed that more than just the first sentence usually made sense, but also that the chatbot often repeated its answer. Thus, we defined a method to return the N first sentences in the output.

3.2 GRU Model

GRU has a less complex structure compared to the LSTM and is computationally more efficient. We want to explore how well the GRU conversational agents perform, and if the time required to train a GRU model is significantly less than the training of the baseline. TensorFlow's Sequence-to-Sequence interface is motivated by the many different models proposed in recent papers and makes it easy to experiment with both LSTM and GRU cells, despite their architectural differences. To change from LSTM to GRU cells in the LSTM baseline model, we modify the cell as demonstrated in Listing 3.5.

```
    single_cell = tf.nn.rnn_cell.GRUCell(size)
    cell = single_cell
    if num_layers > 1:
        cell = tf.nn.rnn_cell.MultiRNNCell([single_cell] * num_layers)
```

Listing 3.5: Create LSTM model

3.3 Grid LSTM Model

Our goal is to use the Grid LSTM cell, as proposed in Kalchbrenner et al. [2015], to experiment with the state-of-the-art in the field of LSTM architectures. The use of the Grid LSTM cell in a chatbot is motivated by the challenging task of representing all of the information in a sentence in one vector. The attention mechanism, mentioned in Section 2.2, relieves some of the pressure on the encoder, which the Grid LSTM further improves as a result of the structure of the Grid LSTM block. The input is not compressed to one vector, but each word in the input sequence is projected on one side of the grid. Thus, the model can repeatedly scan the source sentence. Therefore, we want to see whether or not this has an effect on the quality of the output from a conversational agent.

We relate our problem to the translation experiment described by Kalchbrenner et al. [2015], illustrated in Figure 3.3. This architecture is a two layered 3-LSTM blocks design, meaning that each layer is a grid construction of 3-LSTM blocks. It appears like two 2-LSTMs stacked on top of each other, but each block is connected to the corresponding block in the other layer. However, during the test phases, we noticed that the model's loss decreased very slowly with the use of 3-LSTM block. By experimenting with the 2-LSTM block in the same environment, we observed a quickly decreasing loss. Hence, we proceeded with the 2-LSTM block instead of the 3-LSTM block.

The model is comparable to the baseline, with the change of cells and a bidirectional processing of the data. Furthermore, as the dimensions of the cells increase, the model needs to handle the computational growth. We adopt Tensorflow's Contrib[3] module for constructing GridRNN cells[4], and further modified the code to give the model the same properties as the translation model in Kalchbrenner et al. [2015]. An example of the creation of the Grid LSTM model is shown in Listing 3.6. One change we had to apply to the Grid2LSTMCell class was that the weights should be tied as they are shared across both the source and the target sentence. The translation experiment uses this to capture reordering of patterns in sentence structure across different languages. Transferred to the conversational agent problem, the intention with the tied weights is to capture the source of the problem in the source sentence and the source of the solution in the corresponding answer. To match the translation example, the model must process the source sentence and produce the target in different dimensions. Instead of the MultiRNNCell class[5], we create a new Bidirectional class to construct the two layers.

```
    single_cell = Grid2LSTMCell(size, ...)
    cell = single_cell
    if num_layers == 2:
        cell = Bidirectional([single_cell] * num_layers)
```

Listing 3.6: Create Grid LSTM model

[3]TensorFlow Contrib is a directory which is not officially supported, and the code may change or be removed at any time without notice.

[4]https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/grid_rnn/python/ops/grid_rnn_cell.py

[5]https://www.tensorflow.org/versions/r0.11/api_docs/python/rnn_cell/

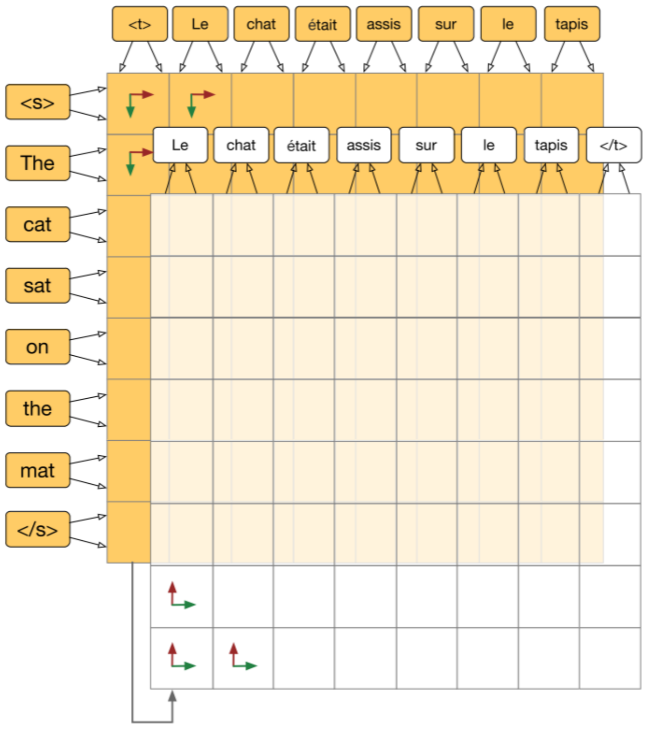


Figure 3.3: Translation model described in Kalchbrenner et al. [2015], consisting of 3-D Blocks.

The layers behave in a bidirectional way, where the first layer scans the source sentence in one direction and the second in the opposite. This design is used to capture the relations between both the preceding and succeeding words, and thus increase the ability to learn longer sequences of words. Unlike the traditional bidirectional network, where the forward and backward states are computed in parallel and then concatenated [Schuster and Paliwal, 1997], the bidirectional Grid LSTM network will compute the forward states first, and then reverse these states as input to the second layer. A code snippet from the Bidirectional class is presented in Listing 3.7.

```
    cur_inp = inputs
    hidden_output = []

    # Forward processing
    fwd_m_out_list = []
    fwd_cell = self._cells[0]

    cur_state = state[0]
    fwd_m_out, fwd_state_out = fwd_cell(cur_inp, cur_state)
    fwd_m_out_list.append(fwd_m_out)
    hidden_output.append(fwd_state_out)

    # Backward processing
    bwd_m_out_list = []
    bwd_cell = self._cells[1]
    cur_reversed_inputs = array_ops.reverse(cur_inp, [True, False])
    cur_state = state[1]
    bwd_m_out, bwd_state_out = bwd_cell(cur_reversed_inputs, cur_state
    )
    bwd_m_out_list.append(bwd_m_out)

    hidden_output.append(bwd_state_out)

    hidden_output = tuple(hidden_output)

    outputs_fwd = nest.pack_sequence_as(structure=fwd_m_out,
    flat_sequence=fwd_m_out_list)
    outputs_bwd = nest.pack_sequence_as(structure=bwd_m_out,
    flat_sequence=bwd_m_out_list)

    memory_output = array_ops.concat(1, outputs_fwd + outputs_bwd)
    return memory_output, hidden_output
```

Listing 3.7: Bidirectional processing

3.4 Stateful Model

The previous architectures do not handle information from preceding turns in a conversation, but simply returns the most predicted answer to the current question. This can be problematic as a question such as *"How do I do that?"* needs a context to make sense. In order to mimic a human conversation, we need to look at prior turns to give the chatbot the ability to reuse this information. The goal is a conversational agent that can use the context of the conversation, even though the most recent question does not contain any information about the topic.

The idea behind the Stateful model is to use the previously generated state in the RNN decoder as the initial state for the next turn. The decoder will, therefore, be adjusted to the previously generated information, when the next question arises. It is reasonable that some of the contexts from the conversation will be passed on, and that the chatbot can use this information to answer follow-up questions.

The encoder in the Stateful model is identical to the encoder in the LSTM baseline. The RNN decoder, however, required some adjustments. Figure 3.4 sketches the processing of four training pairs for the Stateful Model. When the model processes training pairs from the same conversation, it will return the decoder state for every training step and feed it into a Tensorflow placeholder[6]. The next step will further use this LSTM state. When a conversation in our training data ends, we reset the state to make sure that no context is swapped between two different conversations. The code example in Listing 3.8 illustrates that the state is handled when a new batch is filled, and that each training step returns the current state from the model. The *get_stateful_batch* keeps control of all of the conversations and will also reset the state if necessarily.

```
state = initial_state
while max_train_steps >= current_step:
    train_set, batch_train_set, state = get_stateful_batch(train_data, state, ...)
    encoder_inputs, decoder_inputs, target_weights = model.get_batch(batch_train_set)
    _, step_loss, _, state = model.step(encoder_inputs, decoder_inputs, target_weights, state, ...)
```

Listing 3.8: Stateful training

In Section 2.1.5, we described the advantage of training in batches. The use of batches is trivial in the case of a non-stateful model, as the training steps are independent of each other. For the Stateful model, we need to connect the training steps as we will pass the decoder state to the next input. Within a batch, however, each training pair is trained independently, meaning that every batch obtains

[6]https://www.tensorflow.org/api_docs/python/tf/placeholder

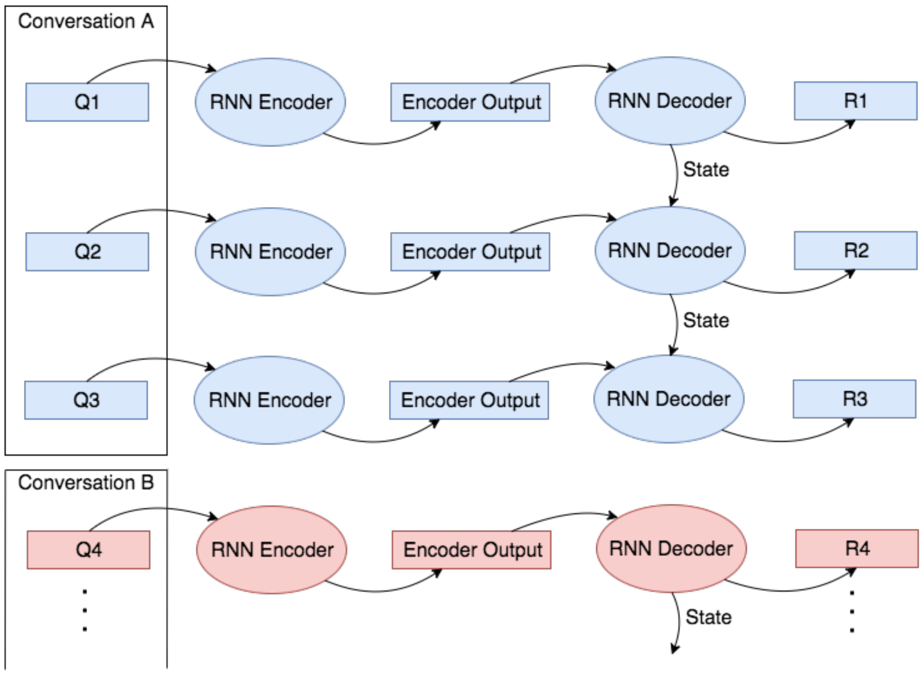


Figure 3.4: Stateful training. The decoder state will be passed on as an initial decoder state for the next training step, if the next training step belongs to the same conversation. At the start of a new conversation, a new zero-initialized LSTM state is created.

multiple decoder states. In other words, with a batch of size N, we can train N conversations simultaneously. Because of this, and the fact that the conversations cannot be mixed, we need N conversation holders as batch feeders. Each batch feeder pays attention to a single conversation at the time and feeds each batch with one training pair from the current conversation. When any of these feeders goes empty, a new conversation is read from the training file and tells the model that it should reset the state for this feeder. This somewhat complicated procedure is illustrated in Figure 3.5, with two batches for simplicity. We, however, used a batch size of 24 in our model to represent a bigger fraction of the data in each step. As a consequence of the stateful training, we had to eliminate the use of buckets for the Stateful model. The reason for this is that the previous state should have an impact on the next sentence in a conversation, regardless of the length of it. If we were using different buckets, this would be impossible.

To our knowledge, this is the first application of an RNN model used for a conversational agent which is trained in a stateful manner.

3.5 Stateful-Decoder

For our context models (Context-Prepro, Stateful Model, Stateful-Decoder), we only output the first sentence from the decoder. This is done to avoid confusion to the user and the model, as we have no guarantee for the quality of the previous responses. A stateful decoder will, just as for the stateful training, pass on the decoder states during the conversation. In Section 3.1 we presented a simple code for the baseline (Listing 3.4). In Listing 3.9, the only difference is the reuse of the state.

```
model = create_model(forward_pass=true, ...)
model.batch_size = 1
state = initial_state
while interacting:
    encoder_inputs, decoder_inputs, target_weights = model.get_batch(user_input, ...)
    _, _, output, state = model.step(encoder_inputs, decoder_inputs, target_weights, state, forward_pass=true, ...)
```

Listing 3.9: Stateful decoding

The Stateful-Decoder model is identical to the LSTM baseline with one bucket during training, but during decoding, it is identical to the Stateful model.

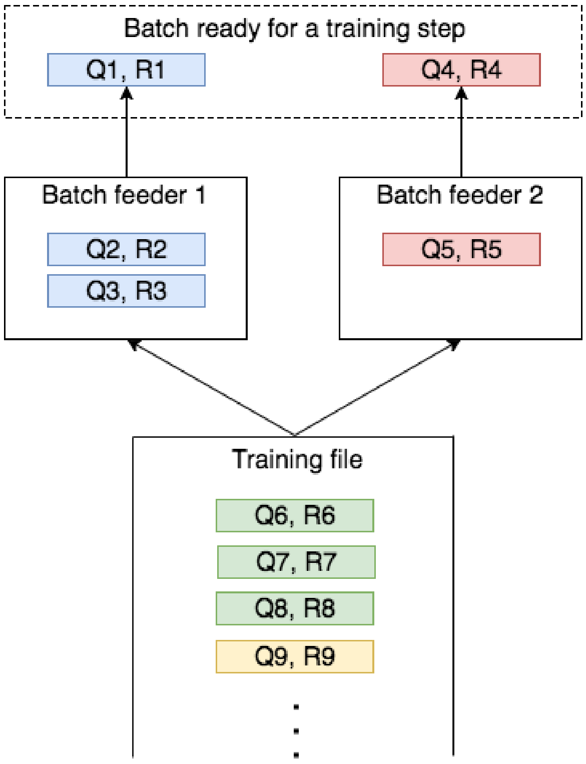


Figure 3.5: Stateful batch feeding. The training file provides the batch holders with conversations. At once one of the holders gets empty, a new conversation is fed from the training file. The feeders continuously supply the corresponding batch with exactly one new training pair for each step to ensure that the conversations do not mix up. Identical color implies that the training pair QN, RN belongs to the same conversation. Training pair $(Q1, R1)$ and $(Q4, R4)$ are trained independently. However, the LSTM state from $(Q1, R1)$ is reused in the decoder when the model handles $(Q2, R2)$. This is also the case for $(Q4, R4)$ and $(Q5, R5)$.

3.6 Context-Prepro

The only difference between this model and the baseline, is that the input will consist of the previous response and the current question. Thus, we need to concatenate these sentences before sending it in to the encoder shown in Listing 3.10. The preprocessing of the training data for this model is further explained in Section 5.1.3. We refer to this approach as the Context-Prepro model.

```
    model = create_model(forward_pass=true, ...)
    model.batch_size = 1
    output = "" # Empty for the first turn
    while interacting:
        question = preprocess(output + user_input, ...)
        encoder_inputs, decoder_inputs, target_weights = model.
    get_batch(question, ...)
        _, _, output = model.step(encoder_inputs, decoder_inputs,
    target_weights, forward_pass=true, ...)
```

Listing 3.10: Context-Prepro decoding

Chapter 4

Datasets

Digital communication has changed the way humans interact, which opens opportunities to collect quantitative datasets. However, it is still a challenge to find well-suited datasets for conversational agents due to noise, which is characterized as data that has an entirely different pattern or structure than rest of the dataset. Reddit comments[1], Twitter[2] and Ubuntu Dialogue Corpus[3] (UDC) are all examples of open source datasets that are continuously growing and available for everyone. This chapter studies the datasets used for the conversational agents presented in this thesis, and focus on the structure, characteristics, and shortcomings of each dataset. Section 4.1 describes the UDC, whereas Section 4.2 take a closer look at the OpenSubtitles dataset.

4.1 Ubuntu Dialogue Corpus

The UDC is a closed domain dataset based on the Ubuntu Chat Logs[4], which is a forum used to discuss technical issues concerning the Ubuntu operative system. This can be used to train a conversational agent within the branch of technical support.

The script from the Ubuntu Dataset Creator[5] will download and store the conversations in tsv (tab separated values) files, each conversation in a distinct file. An example of such a file is depicted in Table 4.1. Each line consists of a

[1]https://www.reddit.com/r/datasets/comments/3bxlg7/i_have_every_publicly_available_ reddit_comment/

[2]https://dev.twitter.com/overview/api

[3]https://github.com/rkadlec/ubuntu-ranking-dataset-creator

[4]https://irclogs.ubuntu.com/

[5]https://github.com/rkadlec/ubuntu-ranking-dataset-creator

timestamp indicating when the message was sent, the username of the person who sent the message, the username of the receiver and the message itself. The parlance used in forums often consists of slang words and abbreviations, considering the low threshold for participating in discussions. Moreover, the UDC consists of directory paths, emoticons, terminal commands, and URLs. Some of these elements are used in the conversations in Table 4.2 and 4.3. More examples are listed in Appendix A.

Timestamp	**Sender**	**Receiver**	**Message**
2005-05-25T11:26:00.000Z	seth		Also guys, I'm trying to get into my Firefox preferences, but it keeps telling me "root.disabled" = "true"
2005-05-25T11:27:00.000Z	lifeless	seth	are you logged in as 'root' ?
2005-05-25T11:27:00.000Z	seth	lifeless	no

Table 4.1: UDC data format example

The advantage of the UDC is the enormous amount of data that is available. When extracting conversations with two or more turns (where one turn is a sequence of consecutive messages from the same user), we get an initial dataset consisting of 8.6M turns built upon 168.7M words. Table 4.4 illustrates the average number of words in a turn and the turn mode, i.e. the most common length of a turn, in the input and output data. Having a dataset of this size gives us the opportunity to remove data that does not fulfill certain requirements, such as reply lengths and questions without responses. Even though anyone can participate in online forums, the sentence structure is fairly good.

However, the low threshold for participating also leads to an informal and oral language. The fact that we have no insurance about the content of the data means that this dataset also suffers from noisy data. The response to a question may differ from other datasets, as an URL may be a valid response to a question if it navigates to a correct answer. However, if the response only consists of an URL, it is difficult to say anything about the chatbot's ability to produce a well-structured sentence. Table 4.2 illustrates another problem, where a user (in this case "ajavid") may type several consecutive responses. This leads to long turns, which are more challenging to handle for a conversational agent.

Jackrabbit:	can anyone help me?
ajavid:	ask
Jackrabbit:	its a program that gets ati video drivers becuase right now my screen is max lol
ajavid:	lspci|grep VGA
ajavid:	try to use packaged software in ubuntu as much as possible
Jackrabbit:	ok so what do I do with that command do I put it in terminal?
ajavid:	r5xx and below on ati == free 3d accel with xorg 7.4 and latest mesa in jaunty
ajavid:	for r6xx + only do you require the fglrx driver
ajavid:	whate exactly are you trying to do?
ajavid:	don't do random things like that
ajavid:	in console terminal, sudo aptitude pciutils; lspci|grep VGA
Jackrabbit:	ok is that all on line like: sudo aptitude pciutils; lspci|grep VGA
ajavid:	yes, ; is a bash newcommand delimiter
Jackrabbit:	ahhhh man I tell ya once my screen is fixed I cant wait to read and learn about this OS becuase I played with

Table 4.2: UDC content example 1

4.2 OpenSubtitles

The OpenSubtitles[6] dataset consist of conversations from movie manuscripts. This dataset is categorized as an open domain dataset because there are no restrictions on topics in film manuscripts. Thus, we find it interesting to see if a chatbot trained on this dataset will do well at the chit-chatting task.

We used the OpenSubtitles-Parser[7] to download the English manuscripts. This gave us 2.7M sentences, built of 19M words. As you can see from Table 4.5, the average number of words in a sentence is less than the UDC average. The parser will, in addition to download the dataset, do some preprocessing such as adding spaces after apostrophes. Further, some words were concatenated, e.g. *theysaid*, *youknow*, as if the writer tried to type the manuscript fast. "*l*" and "*i's*" were

[6]http://www.opensubtitles.org/
[7]https://github.com/inikdom/opensubtitles-parser

Siegel-	im pastebining the entire thing.
PatrickDickey	in the terminal, type sudo dpkg --configure -a
Siegel-	ok
Siegel-	im gonna pastebin the result: http://pastebin.com/1UQ2g5EV
PatrickDickey	I haven't forgotten you. ;-) You could try sudo dpkg --remove --pending and it should clear those two updates out. Then you can do a check for updates again, and try them again.
Siegel-	thanks would i need to restart? because they still show in update manager
Siegel-	about a month maybe more
Siegel-	it failed. im gonna pastebin details
PatrickDickey	you could try sudo apt-get -f dist-upgrade it's supposed to force things.
Siegel-	im running it

Table 4.3: UDC content example 2

	Input data	**Output data**
Average	17	14
Mode	2	2

Table 4.4: Average turn length and mode turn length

	Input data	**Output data**
Average	7	7
Mode	2	2

Table 4.5: Average turn length and mode turn length

switched in some words, e.g. $caii - call$, $l'm - I'm$, $reaily - really$. Actions or other descriptions were written in brackets, e.g. [*song*], [*cries*]. We also noticed manuscripts where XML symbols, e.g. $< br >$, $< i >$, were present.

```
For Your Eyes Only, 1981
good afternoon , mr.  bond .
don' t concern yourself with the pilot .
one of my less useful people .
you are now flying remote control airways .
think twice , 007 .
it' s a long way down .
i' ve looked forward to this moment , mr.  bond .
i intend to enjoy it to the full .
really , have you no respect for the dead ?
good bye , mr.  bond .
End of Days, 1999
life keeps on tickin ' , tickin ' , tickin ' , tickin '
[ song fades out ]
what' s wrong , baby ?
what' s wrong ?
it was a dream .
he came for me again .
it' s only a dream , angel .
it felt closer .
through out the country , the national guard ...  has
been put on alert as a precaution [ male newscaster ]
mayor giulianiurged all citizens and police ...  to stay
calm during this holiday season .
```

no explanations for how blood on the hands of the christ child , depictedin michelangelo’ s famous sculpture of the pieta [male newscaster]

Gladiator, 2000

i served with you at vindobona .
you can help me .
whatever comes out of these gates ... we’ ve got a better chance of survival if we work together .
do you understand ?
if we stay together , we survive .
the emperor is pleased to bring you the legionnaires ...
of scipio africanus !
to the death !
kill !
kill !
kill !

Bad Company, 2002

youknow , poor people do get married .
i know , but i don’ t feel like fighting with you ...
every day because we’ re broke .
marriage is hard enough without being poor .
hey , what happened to the love ?
what happened to all that romance ?
youknow i love you .
but youare living in this fantasy .
it’ s like you’ re waiting for some kind of miracle .
i gotta live in reality .
i am not gonna be young forever .

Fun with Dick and Jane, 2005

hey , veronica .
hi , how are you ?
good , good .
i didn’ t know you worked out here .
welcome to kostmart .
i hope you’ il take a trip by the deli today ... ...
for a complimentary cube of jalapeno cheddar .
i’ m lactose intolerant .
where do you keep the cigarettes ?
behind the counter .
but i’ m not sure that’ s a good ...
dick ?

```
you missed one .
```

Table 4.6: OpenSubtitles content example

Because OpenSubtitles is based on movie subtitles, we have some assurance of grammatical content. On the other hand, there are two main issues with this dataset. First, it is hard to know whether or not two replies share context, as the manuscripts are not divided into the different scenes. Second, we do not know if the following sentence is a reply to the previous sentence, or if the same person speaks twice. The manuscript from Table 4.6 reflects the difficulties we encounter when training models on this dataset. The main problem is that it is hard to extract good question and response pairs. Unlike the UDC dataset, we are not provided with any information about which actor who says what. More examples of the dataset are listed in Appendix B.

Chapter 5

Experimental Settings

This chapter presents the experimental part of the research project, and describes how we prepared, measured, planned, and conducted the experiments. Section 5.1 defines four preprocessing steps, and further the adjustments required in these steps for the different datasets and the context-based models. Then, the metrics used for evaluating the models are presented in Section 5.2. Section 5.3 outlines the experimental plan. It also contains all the information required to repeat the experiments and explains how to interact with the agents. The hardware used for training is a distributed GeForce GTX TITAN X GPU, with standard memory config at 12GB and a memory bandwidth of 336.5 GB/sec, 3072 CUDA cores and a base clock of 1000 MHz. Our code is written in Python 2.7, applied with Tensorflow[1], an open-source software library for Machine Intelligence.

5.1 Preprocessing

In this section, we describe the preprocessing steps, from the initial dataset to the training data consisting of question-response pairs. Further, we explain the adjustments required in the preprocessing script, when creating training data from the UDC and OpenSubtitles datasets. Similarly, the context-based models also need the data in a certain way, and these changes are further described.

5.1.1 Preprocessing Overview

An overview of the preprocessing is depicted in Figure 5.1. During this process, we save the questions in input files, and the responses in output files, referred to as I and O in the figures. We cannot teach a chatbot to speak using an infinite

[1]https://www.tensorflow.org/

vocabulary, as training and decoding complexity increase proportionally to the number of target words. Depending on the dataset, we choose a proper vocabulary size consisting of the X most frequent words, while the rest of the words will be categorized as out-of-vocabulary (OOV) words. Handling these OOV words is typically done in two ways: 1) represent them with an unknown token or 2) remove them from the dataset. The former solution is most common, but has one major drawback: the chatbot might output an unknown token which does not contribute to the sentence's meaning. The latter option will destroy the structure of the sentence. This section introduces an improved solution, where the idea is to replace every OOV word with the most similar word in the vocabulary based on both semantic and morphology. This approach is described in Step 2 and 3. The preprocessing can easily be adjusted to be used by other datasets. Simply change the extract part in Step 1, and optionally adjust the regular expressions or "word replacements" that should be performed. A regular expression, or regex, defines a search pattern used to find or replace strings. The remainder of this section describes each preprocessing step in detail.

Step 1: Extract dialogs

The first step is to extract the question-response pairs from the dataset, and is illustrated in Figure 5.2. The questions are saved in the input file, and the responses are saved in the output file. Given an index x in the input file, the response appears at line x in the output file. Depending on the characteristics of the dataset, we do actions to certain words, e.g. URLs, abbreviations or words that are concatenated. Then, we check if any of the words are in our "common misspelling file" (CMF), which is based on Normalizer's spell fix file[2]. The file consists of two words on each line, the commonly misspelled word, and the correct word. During this step, the script will iterate through all the words in the dataset and replace the possible CMF-word with the correct one. The reason for why the preprocessing should involve regex and misspelling checks is that the vocabulary should represent as many words as possible. If two different words represent the same semantics, they should not hold two places in the vocabulary, but rather be merged to one. Table 5.2 shows the reduction of unique words for the UDC dataset, and indicates that the vocabulary will cover more of the words occurring in the dataset after this process.

Step 2: Create Word Representation Model

Figure 5.3 shows the second step in the preprocessing procedure. In order to remove the rest of the OOV words, we use the fastText[3] library mentioned in Section 2.2. This library allows us to create our own word representation model, and train it on the data from the previous step. The model will be able to give all words a word embedding which gives the word a position in an n-dimensional

[2]https://github.com/superscriptjs/normalizer/blob/master/data/spellfix.txt
[3]https://github.com/facebookresearch/fastText

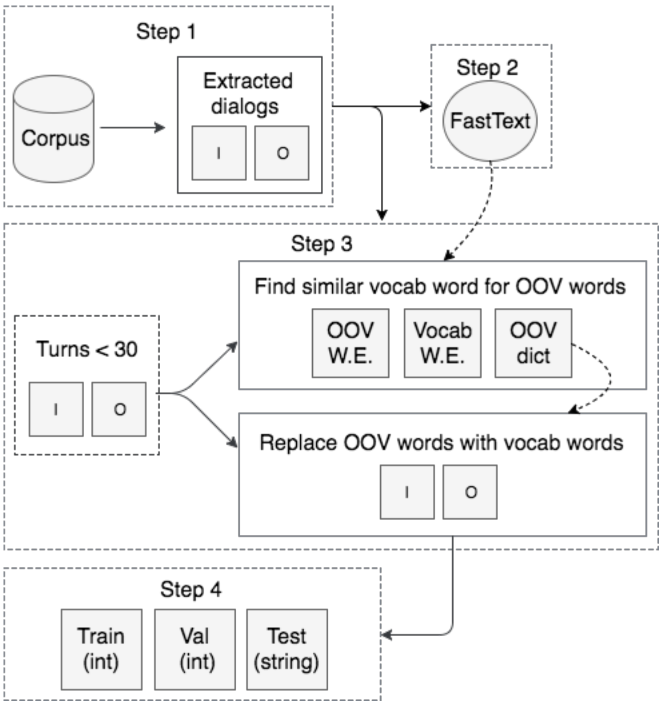


Figure 5.1: Preprocessing - Overview

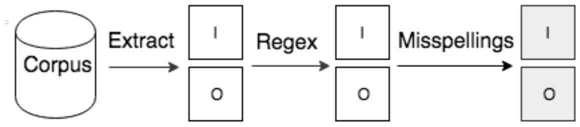


Figure 5.2: Preprocessing - Step 1

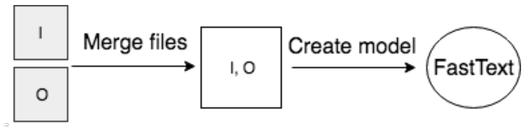


Figure 5.3: Preprocessing - Step 2

space. Even though the dimension size should increase when the dataset is bigger [Bojanowski et al., 2016], we decided to stick to the default 100-dimensional vector because larger vectors are time-consuming to compare. FastText helps us in the process of eliminating all OOV words, as it can generate vector representations to all words which we further use to replace unknown words.

Step 3: Skip long turns and replace OOV words
Figure 5.4 depicts the third step. We set an upper limit to 30 words for the turn

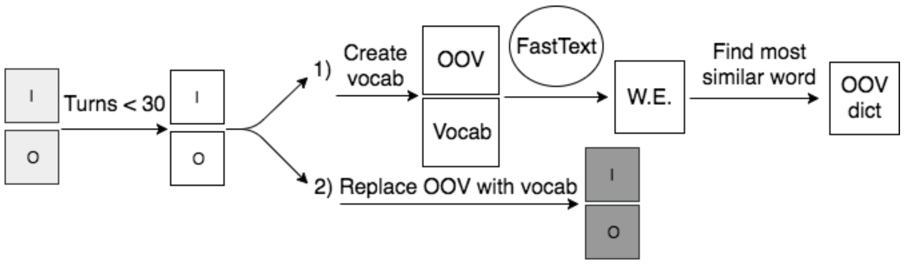


Figure 5.4: Preprocessing - Step 3

length to decrease the preprocessing time and to make the training easier, as it is challenging for chatbots to handle longer sentences. The reason for why we slice the data in this step, and not in the previous steps, is because we wanted to see the effect of the regex and word replacements on the entire corpus. These changes may also affect the turn length of a training pair, to either fit or not fit a bucket.

OOV Word	**Word in vocabulary**
luanch	launch
mailsoerver	mailserver
,meaning	meaning
courcecode	sourcecode
externen	extern
pref->	preferences->
(direction	direction
hey**	hey
??python	python
tomatto	tomato
248mb	96mb
alt+left-drag	alt+left
urprised	surprised
s4487	memtotal:
iglooftp-pro	filezilla

Table 5.1: Examples of replacements of OOV word by calculating the cosine similarity of word embeddings from the word representation model. Typos are recognized and handled in a great way, while rare words sometimes are replaced with something completely different. Further, numbers are switched, which tells us that a special token for integers in the vocabulary should be considered.

Also, the fastText model will perform better by having more training data. Since all words should be represented as an integer in the final training data, we create a vocabulary consisting of the X most frequent words. The vocabulary size will vary depending on the dataset.

A word in the dataset will either be in the vocabulary or be an OOV word. We use the fastText model created in step 2 to find word embeddings for the vocabulary and the OOV words. With all words represented as 100-dimensional vectors, we can calculate the similarity by exploring the embedded space and locate the nearest neighbor to the current word. One approach to do this is by calculating the cosine similarity[4] between every vocabulary words' vector and the OOV word's vector. The best match will be the word which has the smallest angular difference. After this, we replace the OOV word with the most similar vocabulary word. Now, we have two files; input data consisting of questions, and output data with responses. These files will only consist of words from the vocabulary. Thus, we have eliminated the problems with unknown tokens. Figure 5.1 shows examples of OOV that is replaced by the most similar vocabulary word.

[4]https://en.wikipedia.org/wiki/Cosine_similarity

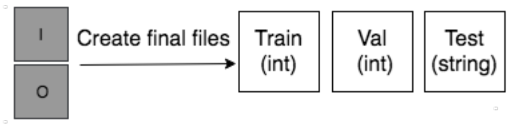


Figure 5.5: Preprocessing - Step 4

Step 4: Create final files
In the final step, depicted in Figure 5.5, we prepare the data for the Encoder-Decoder model, which require the data represented as integers, where the input and output data are separated by a comma. The questions and responses are merged into one file, and the lines are then shuffled. The shuffle helps us to get a random distribution of the dataset in one file. We use a fraction of this file as train, validation and test data. 80% is used for the training, 10% for the validation and 10% for test data. The test set is then converted back to readable text since this is used for human evaluation purposes. The validation data is used for evaluating the model during training with data it has not trained on before.

5.1.2 Preprocessing UDC

A user can send several consecutive messages, as illustrated in the conversations in Table 4.2 and 4.3. In the extracting step, we keep track of which user that speaks and concatenate consecutive sentences from the same user in one turn. We save the utterances from the initial user into the input data and the utterances from the responding user in the output data.

We use regex to recognize emoticons, URLs, and directory paths and replace them with $\_EMJ$, $\_URL$, and $\_DIR$ tokens, respectively. In the dataset, quotation marks are placed in pairs around a word or a phrase, and the model will handle the words in (“hello world”) and (hello world) differently. In most cases, the semantic of a word will not change because of a quotation mark; thus we will remove them. Furthermore, we added other spelling errors we observed in the dataset, e.g. paswrd to password, to the CMF. In addition, we replace abbreviations found in Wikipedia's list of English contractions[5], such as “that's” to “that is”. Table 5.2 depicts how we reduce the number of unique words by doing these changes.

Before deciding the vocabulary size, we studied how large vocabulary that is needed to cover a certain percentage of the dataset. Figure 5.6 shows how many words that is required in the vocabulary to cover 95-99% of the UDC. We decided to have a vocabulary consisting of the 30k most frequent words, which will be able

[5]https://en.wikipedia.org/wiki/Wikipedia:List_of_English_contractions

Step	**Initial dataset**	**Regex**	**Misspellings**
Unique words	2 451 308	1 320 772	1 318 619

Table 5.2: Unique words in the dataset after each step

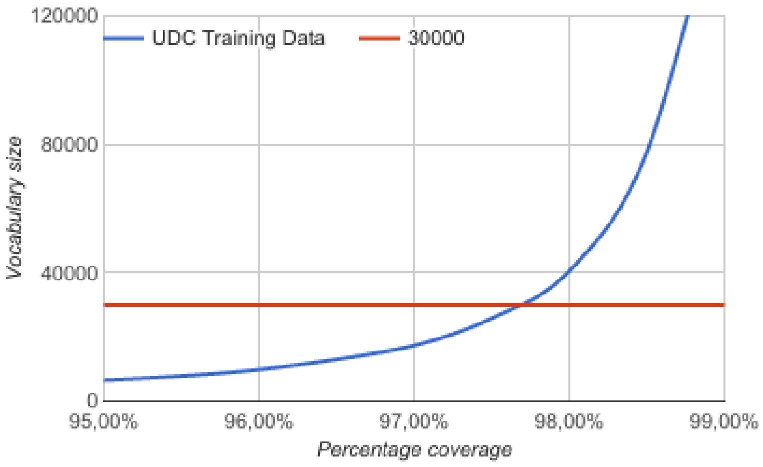


Figure 5.6: Vocabulary size needed to cover [95,99] % of the dataset. The blue line describes the vocabulary size needed to cover a percentage of the dataset.

to represent 97,7% of the dataset. The remaining 2,3% is categorized as OOV words and will be replaced with the words in the vocabulary during Step 3.

The buckets used in the model, should each cover a decent amount of the training data, but also avoid unnecessarily padding. Table 5.3 depicts how much data each bucket covers after preprocessing Step 3. The tuple (A, B) means that the question cannot consist of more than A words and that the response cannot consist of more than B words in order to fit the bucket. For example, a question-response pair with lengths $(9, 15)$ will be placed in bucket $(16, 16)$, and only in this bucket. Even though the turn mode in both the input and output data is 2, the (10,10) bucket does not contain the majority of the dataset. Despite a question-response pair where the question has less than ten words, we have no guarantee that the response has less than ten words and vice versa. If none of the buckets fit, the pair is excluded.

Figure 5.7 and 5.8 illustrate the "bucket-scenario" just described. Figure 5.7 highlights how many turns in the input and output data that have the length described along the x-axis. Individually, both the input and the output data has a majority of only two words in each turn, but if we look at Figure 5.8, we see that

(10, 10)	**(16, 16)**	**(22, 22)**	**(30, 30)**	**Excluded**
8,83%	14.02%	13.39%	13.61%	50.41%

Table 5.3: Bucket size vs. coverage of the initial dataset. A training pair that is placed in bucket (N, N), will only be placed in this bucket.

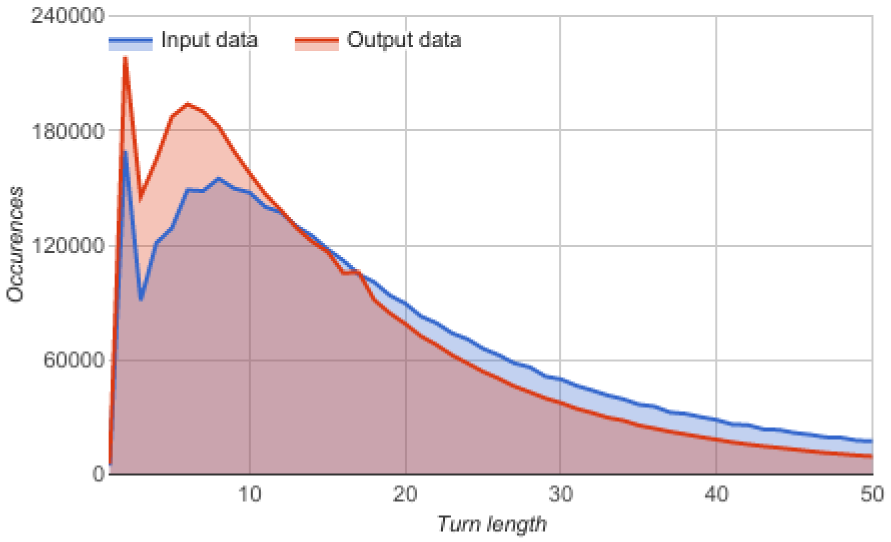


Figure 5.7: Occurrences of turns with length [1,50]

the mode has increased to 14 for the question-response pairs. Based on this, we know that there is a majority of short turns, and besides that the smallest bucket contains less training data, the most of its content has few words. The fact that there is a majority of two-word turns is the reason for why we did not further increase the tiniest bucket. If we increased the size, however, we might cover more data in the bucket, but we would also generate training pairs where more than 80% of the sentences contain $\_PAD$-tokens.

The UDC dataset consists of ~1.8M conversations after step 1, where the average number of turns per conversation is 3.2 turns. When limiting our dataset to only include turns with less than 30 words, the average number of turns per conversation decreases to 2.3 turns.

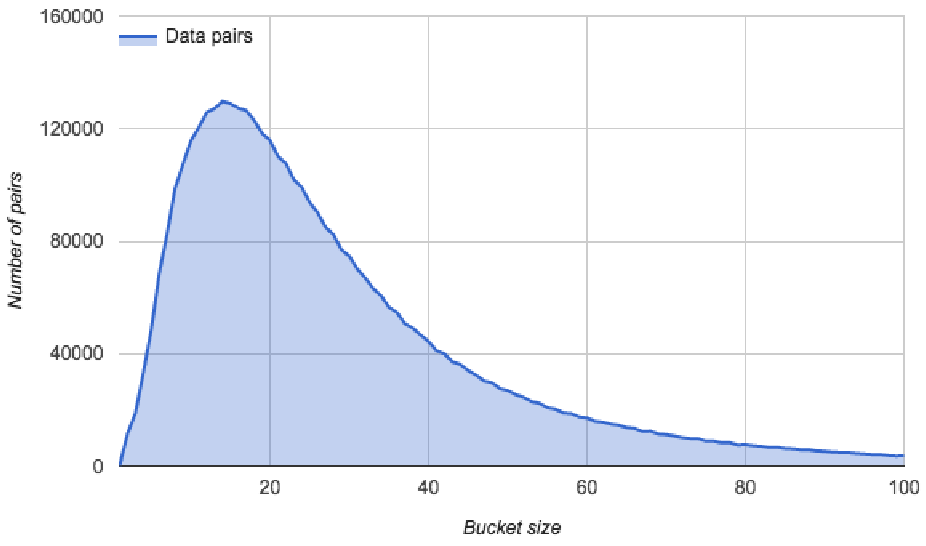


Figure 5.8: Number of pairs that fit the given bucket size. The y-axis describes how many pairs that fits perfectly in the bucket size on the x-axis.

5.1.3 Preprocessing UDC for Context Approaches

In Chapter 3 we described two approaches that did some changes to the model to catch the context of a conversation better, the Stateful and Context-Prepro model. These models require the data further preprocessed and prepared.

Preprocess the Training Data for Context-Prepro

The goal is to include the context in the training data, where the previous answer will be added in front of the next training input, requiring an expansion of the bucket sizes. However, this results in higher memory allocation as seen in Table 2.2, in addition to an increased training time. This naïve approach will indeed carry some of the information from the previous turns.

A similar approach was explored by Sordoni et al. [2015]. The result of their work was a Recurrent Language Model architecture which generated context-sensitive responses. They presented a Tripled Language Model which concatenated the context (previous message), the message itself, and the response into a single sentence s. The network was trained to minimize the negative log-likelihood of the training sentence s.

Instead of just sending in a question, we add some additional information to the

Model	**Training pairs**	**Conversations**
Non-stateful	3648675	1852868*
Stateful	1840502	665393

Table 5.4: Data comparison for non-stateful vs. Stateful.
*The non-stateful model does not use the term conversations. This number is therefore the total amount of conversations in the dataset, not conversations extracted to fit the non-stateful model.

input. Usually, if person A and B talks, the input will have the format $A1 - B1$, $A2 - B2$, ..., $An - Bn$. A1 indicates the first turn in the question data, whereas B1 indicates the first turn in the response data. In this case, we change the input to have the format to $A1 - B1$, $B1A2 - B2$, ..., $B(n-1)An - Bn$. Now, the chatbot has information about its previous answer as well as the question.

There is a trade-off between the amount of information we should concatenate as input to the model and the training complexity. As we have seen, training the model with larger buckets, e.g. longer turns, requires more memory allocation and the training time increases. Thus, we decided to limit the context to the previous answer only.

Preprocess the Training Data for the Stateful Model

The preprocessing for the Stateful model is slightly different from the previously described procedures. So far, the models have not paid attention to the order of the training pairs. The Stateful model, however, will use previous states as inputs when handling the next training pair in a conversation. Thus, the question and response must be listed chronologically. Therefore, the order of the training pairs within a conversation must stay unshuffled, while the conversation itself can be shuffled among the other conversations. As no turns in the conversation can be longer than the predefined bucket size, the preprocessing must filter out all conversations that include turns contradicting this size. Naturally, this restriction removes a lot of the training data, which can be seen in Table 5.4, ending up with ~665k conversations.

For efficiency reasons, we merge all of the conversations into two new training files. While one file is used for training, the other file is shuffled and recreated. An *_EOC*-token separates the conversations within the files.

5.1.4 Preprocessing OpenSubtitles

We do not know whether the following line in the OpenSubtitles dataset is the reply to the previous line or not, and for this reason, each line will be in both the input and output data as Table 5.5 shows. This dataset has other characteristics than

Input data	**Output data**
karl !	over here !
over here !	titus has got something .
titus has got something .	okay , harvey .
okay , harvey .	we have got you now , boy !
we have got you now , boy !	i do not understand .
i do not understand .	what ?
what ?	when we started ... there were two feet .
when we started ... there were two feet .	but now there is four here .
but now there is four here .	these tracks go on for bloody miles .
these tracks go on for bloody miles .	it is bizarre .
it is bizarre .	what are they ?

Table 5.5: Input and output data for OpenSubtitles

the UDC, and multiple words that are not separated by whitespace is a frequent fault. To increase the quality of the training data, we used regex to split words that start with "yours/your/just/why", removed the space after special characters, and removed other characters entirely ($[], <>, @, \#$). We used the same CMF as we described earlier, but added some misspellings frequently observed in the dataset. OpenSubtitles contained originally ~128k unique words, which we reduced by more than ~7k words during this procedure.

Table 5.6 represents the buckets used in the models, whereas Figure 5.9 illustrates the size of the vocabulary needed to cover X% of OpenSubtitles. Having a vocabulary consisting of 20k words cover more than 98% of the dataset.

(6, 6)	**(8, 8)**	**(11, 11)**	**(20, 20)**	**Excluded**
24,13%	22,35%	24,82%	23,70%	5%

Table 5.6: Bucket size vs. coverage of the initial dataset

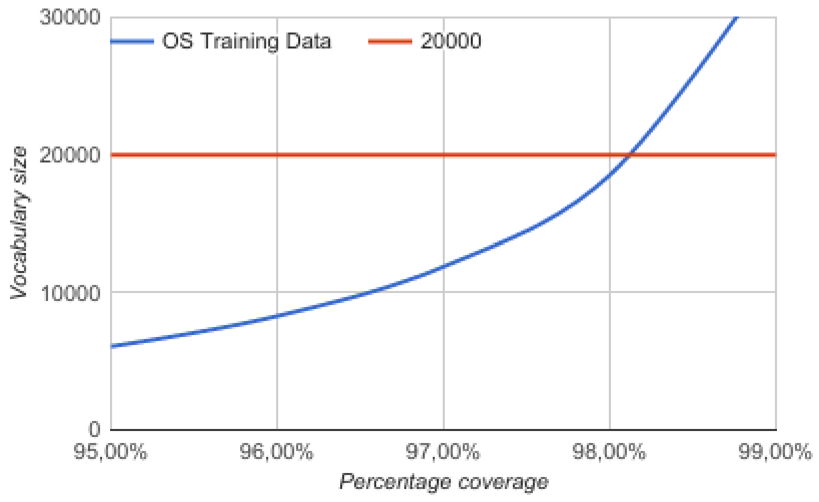


Figure 5.9: Vocabulary size needed to cover [95,99] % of the dataset

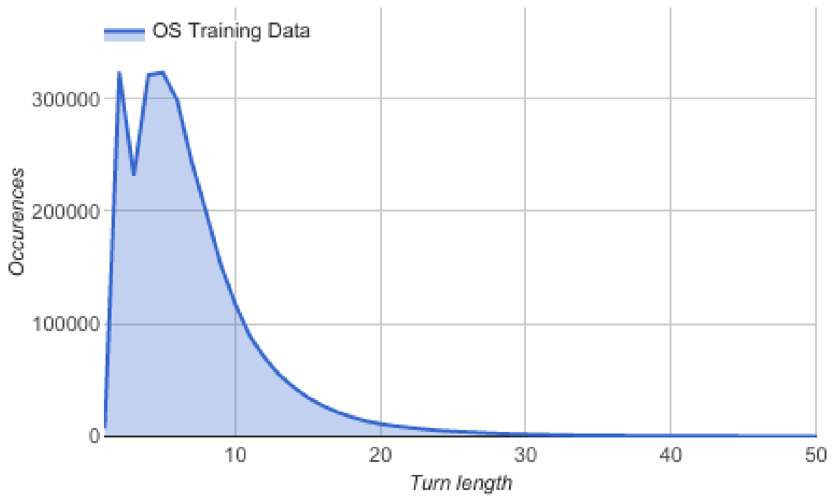


Figure 5.10: Occurrences of turns with length [1,50]

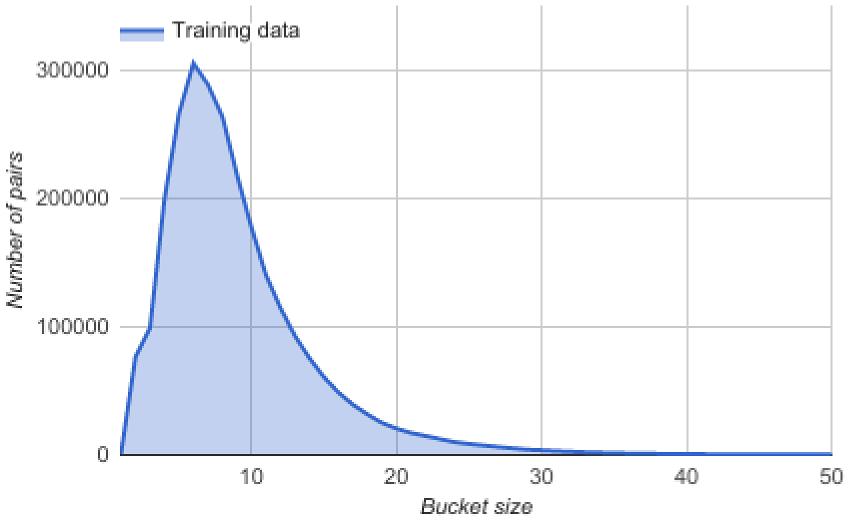


Figure 5.11: Number of pairs that fit the given bucket size

5.2 Metrics

5.2.1 Automatic Evaluation During Training

An automatic evaluation metric for conversational agents is still an open problem and is one of the challenges when working with dialogue systems. However, an automatic feedback during training is required, and we use perplexity as our metric. Perplexity is a commonly used measurement in statistical language modeling, and measure how well a probability model predicts a sample[6]. In our case, a low perplexity means that the model has predicted the response well. The formula for perplexity is expressed in Equation 5.1, where N is the number of training steps between each calculation, and $loss_i$ is the loss in step i.

$$p = exp(\frac{\sum_{i=1}^{N} loss_i}{N}) \tag{5.1}$$

5.2.2 Human Evaluation

We also conduct a Human Evaluation (HE), which evaluates the results from the experiments that will be presented in Section 5.3. The experiments are divided in

[6]https://en.wikipedia.org/wiki/Perplexity

Grammar	**Points**
Very bad grammar	1
Bad grammar	2
Some grammatical errors	3
Good grammar	4
No grammatical errors	5

Table 5.7: Human Evaluation metric - Grammar

Content (Single Questions)	**Points**
Response makes no sense	1
Response makes little sense	2
Response makes some sense	3
Response makes a lot of sense	4
Response makes perfectly sense	5

Table 5.8: Human Evaluation metric - Content (Single questions)

three parts partitioned in two questionnaires. Part 1 and 2 are evaluated in the first questionnaire, while Part 3 is evaluated in the second.

The participants of the HE will receive an information sheet, which explains how the survey is structured, and with guidelines on how to judge the responses. For each question or conversation, the evaluator is asked to rate the answer(s) from the different conversational agents. When Shang et al. [2015] conducted a human evaluation, the participants were asked to rate responses with a score from 0 to 2. We, however, increased the range to be from 1 to 5, based on the feedback from the test persons. When rating the grammar, the evaluators are asked to evaluate the sentence structure and the spelling. When evaluating the content, the participants should ask themselves if the response makes sense and if the chatbot answers the question. When the questions are related, they should expect that the chatbot remembers previous utterances as in a normal human conversation.

Content (Conversations)	**Points**
Only bad responses	1
Many bad responses	2
Mixed quality on the responses	3
Many good responses	4
Only good responses	5

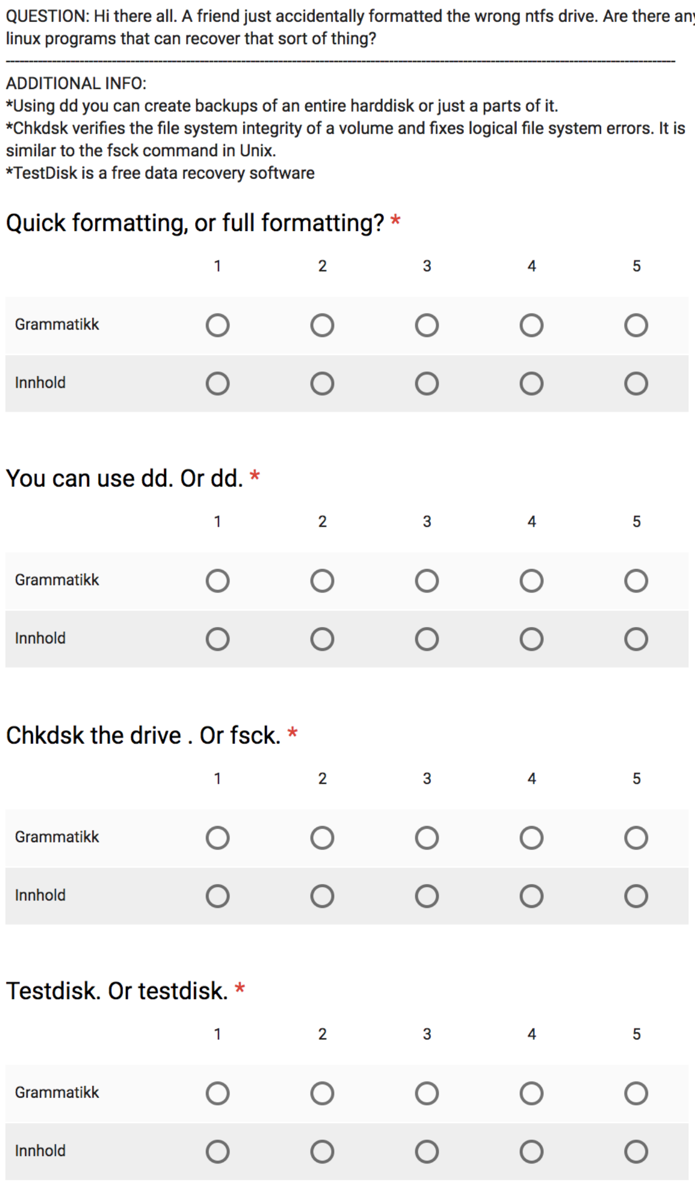
Table 5.9: Human Evaluation metric - Content (Conversations)

HE for Part 1 and 2

The first evaluation aims to answer both research question 1 and 2. Part 1 and 2 of the experiments are evaluated together, as they both concern the UDC dataset. The questions are extracted from this corpus' test set. Recall that UDC is a closed domain dataset which consists of technical Ubuntu related questions and responses. Thus, some of the questions/responses require knowledge about Ubuntu. Due to the technical content, we asked computer science, informatics and communication technology students to participate. 30 students, women and men aged between 23 and 27, participated in this evaluation. However, because not everyone has experience with the Ubuntu operative system, we supplied some of the questions with additional information. Examples of how we presented the question and responses for Part 1 is shown in Figure 5.12, and as you can see, the evaluator is asked to rate single questions. The actual response is listed below each answer, followed by the replies from the chatbots. We chose to include the responses from the test set in the questionnaire, to get an idea of how well the models perform compared to the content in the dataset. An example of the presentation of the conversations from Part 2 is illustrated in Figure 5.13. In this part, the evaluators will rate whole conversations from the chatbots, where the questions may differ as the conversations evolve.

HE for Part 3

The second questionnaire evaluates the results from the conversational agents trained on the OpenSubtitles dataset and is used to get additional feedback for the models to increase the external validity. Considering that the second evaluation does not require any technical knowledge, we did not restrict the participants to tech students. 50 persons, women and men aged between 20 and 64 participated in the evaluation of the final experiment. Due to the problems described in Section 4.2, we did not extract questions, to be used for the questionnaire, from the test set. Instead, we defined seven different topics, and asked the chatbots questions we found suitable within the topics. Based on the feedback from the first evaluation, we tried to shorten the time required to conduct the survey. Therefore, the evaluator is asked to rate the chatbot's responses for that topic, and not each response, even though the questions were identical. Figure 5.14 illustrates this. The questions are listed in the first table, and the answers from each agent are listed in the second table.



QUESTION: Hi there all. A friend just accidentally formatted the wrong ntfs drive. Are there any linux programs that can recover that sort of thing?

ADDITIONAL INFO:
*Using dd you can create backups of an entire harddisk or just a parts of it.
*Chkdsk verifies the file system integrity of a volume and fixes logical file system errors. It is similar to the fsck command in Unix.
*TestDisk is a free data recovery software

Quick formatting, or full formatting? *

	1	2	3	4	5
Grammatikk	○	○	○	○	○
Innhold	○	○	○	○	○

You can use dd. Or dd. *

	1	2	3	4	5
Grammatikk	○	○	○	○	○
Innhold	○	○	○	○	○

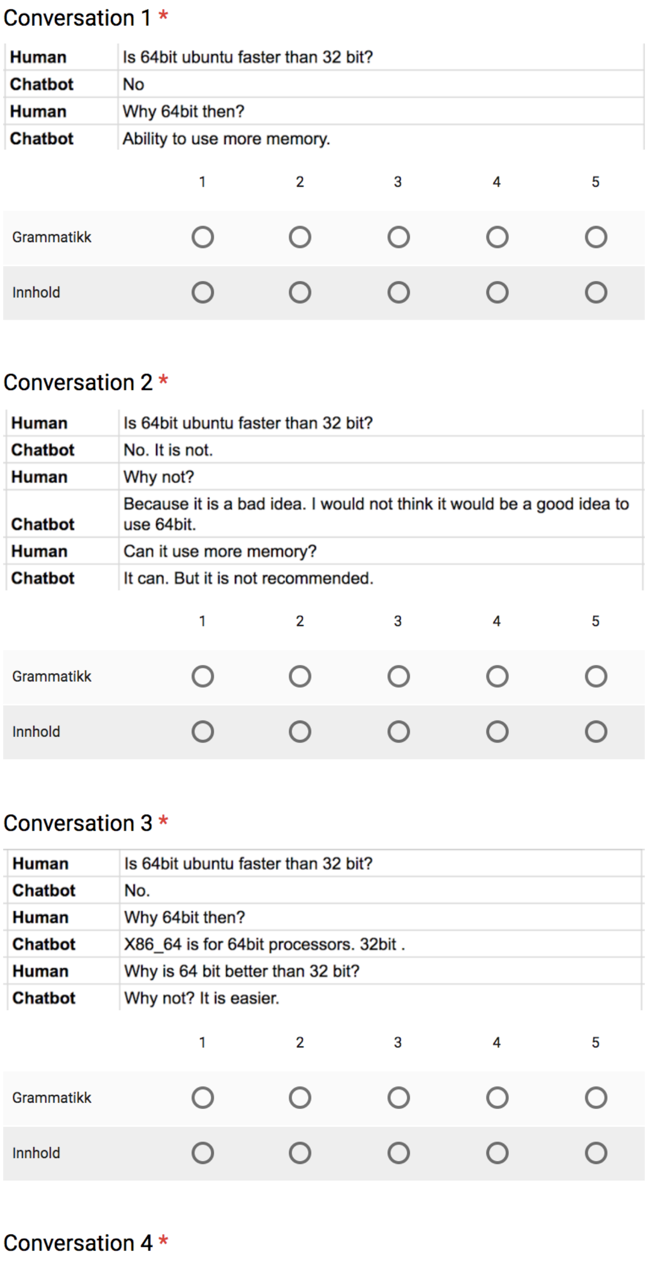
Chkdsk the drive . Or fsck. *

	1	2	3	4	5
Grammatikk	○	○	○	○	○
Innhold	○	○	○	○	○

Testdisk. Or testdisk. *

	1	2	3	4	5
Grammatikk	○	○	○	○	○
Innhold	○	○	○	○	○

Figure 5.12: Screenshot of a question used in the questionnaire for the evaluation of the UDC Part 1 conversational agents



Conversation 1 *

Human	Is 64bit ubuntu faster than 32 bit?
Chatbot	No
Human	Why 64bit then?
Chatbot	Ability to use more memory.

	1	2	3	4	5
Grammatikk	○	○	○	○	○
Innhold	○	○	○	○	○

Conversation 2 *

Human	Is 64bit ubuntu faster than 32 bit?
Chatbot	No. It is not.
Human	Why not?
Chatbot	Because it is a bad idea. I would not think it would be a good idea to use 64bit.
Human	Can it use more memory?
Chatbot	It can. But it is not recommended.

	1	2	3	4	5
Grammatikk	○	○	○	○	○
Innhold	○	○	○	○	○

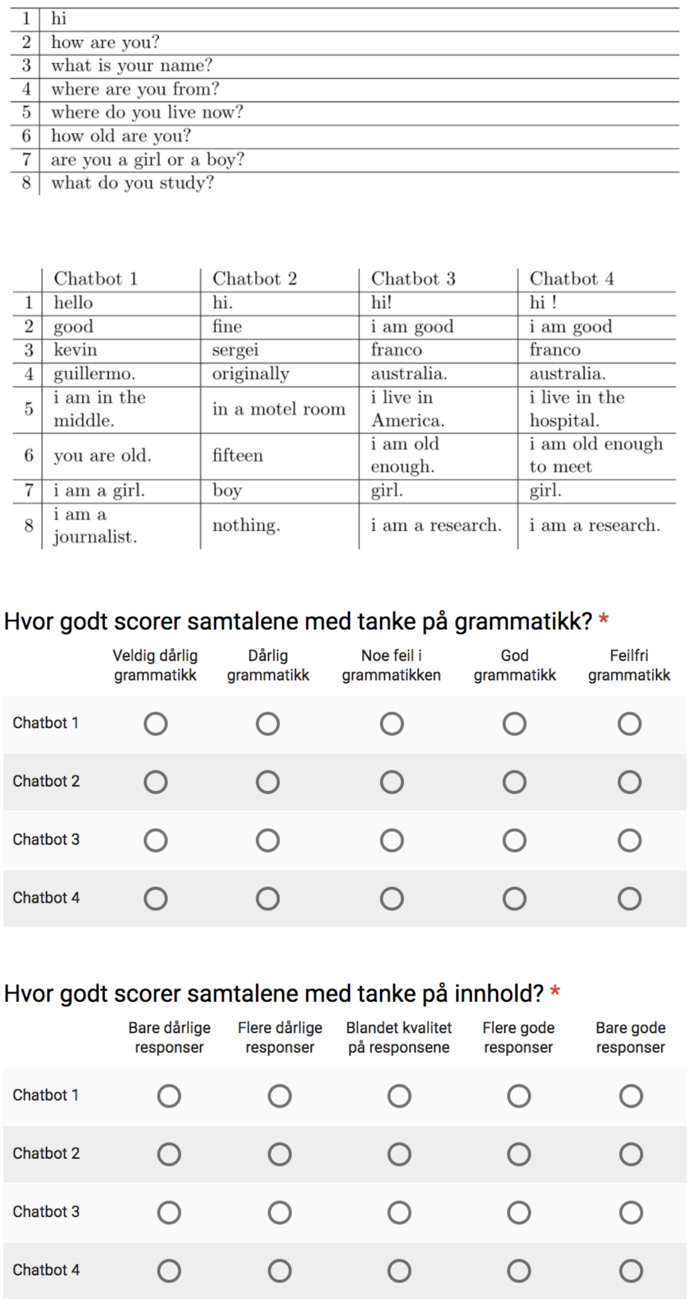
Conversation 3 *

Human	Is 64bit ubuntu faster than 32 bit?
Chatbot	No.
Human	Why 64bit then?
Chatbot	X86_64 is for 64bit processors. 32bit .
Human	Why is 64 bit better than 32 bit?
Chatbot	Why not? It is easier.

	1	2	3	4	5
Grammatikk	○	○	○	○	○
Innhold	○	○	○	○	○

Conversation 4 *

Figure 5.13: Screenshot of a question used in the questionnaire for the evaluation of the UDC Part 2 conversational agents



1	hi
2	how are you?
3	what is your name?
4	where are you from?
5	where do you live now?
6	how old are you?
7	are you a girl or a boy?
8	what do you study?

	Chatbot 1	Chatbot 2	Chatbot 3	Chatbot 4
1	hello	hi.	hi!	hi !
2	good	fine	i am good	i am good
3	kevin	sergei	franco	franco
4	guillermo.	originally	australia.	australia.
5	i am in the middle.	in a motel room	i live in America.	i live in the hospital.
6	you are old.	fifteen	i am old enough.	i am old enough to meet
7	i am a girl.	boy	girl.	girl.
8	i am a journalist.	nothing.	i am a research.	i am a research.

Hvor godt scorer samtalene med tanke på grammatikk? *

	Veldig dårlig grammatikk	Dårlig grammatikk	Noe feil i grammatikken	God grammatikk	Feilfri grammatikk
Chatbot 1	○	○	○	○	○
Chatbot 2	○	○	○	○	○
Chatbot 3	○	○	○	○	○
Chatbot 4	○	○	○	○	○

Hvor godt scorer samtalene med tanke på innhold? *

	Bare dårlige responser	Flere dårlige responser	Blandet kvalitet på responsene	Flere gode responser	Bare gode responser
Chatbot 1	○	○	○	○	○
Chatbot 2	○	○	○	○	○
Chatbot 3	○	○	○	○	○
Chatbot 4	○	○	○	○	○

Figure 5.14: Screenshot of a question used in the questionnaire for the evaluation of the OpenSubtitles conversational agents

5.3 Experimental Plan and Setup

5.3.1 Preparation for the Experiments

The project can be cloned from our GitHub repository [Christensen and Johnsrud, 2017]. All of the models are found in the "Models" folder, whereas the datafiles should be in "Preprocessing". Before you can start the training, you need to download one of the datasets (UDC[7], OpenSubtitles[8]). These datasets should be located in the same folder as the GitHub repository cloned. To start the preprocessing of the datasets, run the following command to preprocess UDC:

```
python preprocessing.py
```

To preprocess OpenSubtitles, run this command:

```
python preprocessing.py --open_subtitles
```

Recall that for each OOV word, we need to calculate the cosine similarity to all the words in the vocabulary. Therefore, the preprocessing takes several hours as this is a time-consuming process.

Certain parameters will be defined in a variables.py file, while others can be changed by using flags when running a model, which we describe in this section. In variables.py we define the paths to the preprocessed files, the size of the network, which model and dataset to use. Further, it sets the optimizer, batch size, bucket sizes, training steps and how often to calculate and save checkpoints. All of the models trained in this research project have two layers with 1024 units. We applied the AdaGrad optimizer and had a batch size of 24. In the next subsections, we will describe the adjustments to each model, and how to start the training procedure. To interact with a chatbot, simply type the same command used to train the model, in addition to a decode flag. As an example, the following command should be executed to chat with the baseline model trained on the OpenSubtitles dataset:

```
python LSTM.py --open_subtitles --decode
```

5.3.2 Part 1: Testing RNN Cells

This experiment intends to answer the first research question; if the use of different RNN cells in an Encoder-Decoder model will have significantly impact on the quality of the outputs. The purpose of Part 1 is to see whether the Grid LSTM model handles the context in a sentence better than the LSTM and GRU model and if one of the chatbots is superior when it comes to the grammar. We train the models on the dataset described in Subsection 5.1.2.

[7]https://github.com/rkadlec/ubuntu-ranking-dataset-creator
[8]https://github.com/inikdom/opensubtitles-parser

The models are trained for 630k steps. We applied the following bucket sizes: [(10-10),(16-16),(22-22),(30-30)], as described in Section 5.1.2. To train the LSTM, GRU or the Grid LSTM model, simply run one of the following commands:

LSTM

```
python LSTM.py
```

GRU

```
python LSTM.py --use_lstm=false
```

Grid LSTM

```
python GridLSTM.py
```

5.3.3 Part 2: Exploring Context Approaches

In this part, we aim to answer the second research question; if a conversational agent that handles information from previous turns will capture the context of a conversation better. We experiment with two context approaches, using different architectures and different datasets, and compare the proposed models with the LSTM baseline. The training data used for the context experiments are described in Section 5.1.3. The first context approach, the Context-Prepro model, uses training data that contains information from the previous response. During interaction with a user, it concatenates its last response with the human question. This might affect the new response positively if the last response contains useful information, or negatively if the previous reply was inadequate. The second model is the Stateful model described in Section 3.4. We compare both models with the baseline trained on the original preprocessed UDC dataset described in Section 5.1.2

The Context-Prepro and Stateful model are trained for 630k steps. We expanded the input-side of the buckets for the Context-Prepro model because the previous response is concatenated with the question. The buckets will have shape [(20-10),(32-16),(44-22),(60-30)]. For the Stateful model, we obtain the largest bucket: [(30,30)].

Context-Prepro

```
python LSTM.py --context_full_turns
```

Stateful

```
python LSTM_stateful.py
```

5.3.4 Part 3: Increasing the external validity

Part 3 of the experiment will aid us in answering research question 1, in addition to test the stateful decoder to help us answer research question 2. For the first research question, we study how the Grid LSTM model handles chit-chatting compared to the LSTM baseline using OpenSubtiltes as described in Section 4.2. Because a single movie manuscript usually is longer than the normal UDC conversation, and the context changes several times due to different scenes in a movie, we do not train the context models on this dataset. As a compromise, we embed the Stateful Decoder (Section 3.5) to the baseline, to see if the decoder can catch the context of a whole conversation, even though the training procedure did not follow the stateful approach.

The following models were also trained for 630k steps. The bucket sizes are [(6,6),(8,8),(11,11),(20,20)].

LSTM

```
python LSTM.py --open_subtitles
```

Grid LSTM

```
python GridLSTM.py --open_subtitles
```

One-Bucket

Instead of several bucket sizes, this model has only one. We apply the largest bucket (20,20) to make the model capable of answering as long questions as the other models.

```
python LSTM.py --open_subtitles --one_bucket
```

Stateful-Decoder

The Stateful-Decoder requires a single bucket, and hence, we will use the largest bucket (20,20) to make it capable of answering as long questions as the other models.

```
python LSTM.py --open_subtitles --one_bucket
```

This model is therefore identical to the One-Bucket during training. The difference is during decoding:

```
python LSTM.py --open_subtitles --one_bucket --decode --stateful
```

Chapter 6

Results and Discussion

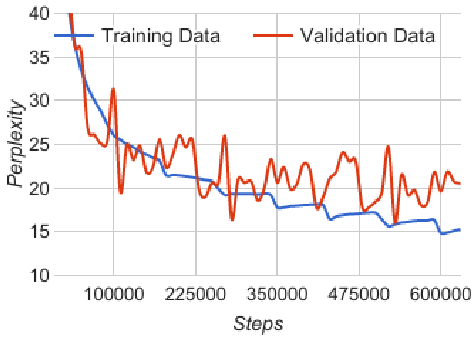
This chapter presents the results of the experiments. First, we will walk through each part of the experiment sequentially, starting with Part 1 in Section 6.1. Then, Part 2 follows in Section 6.2, and finally, Part 3 in Section 6.3. Each Section will first look at the perplexity results before the conversational agents outputs are presented. Then, the Human Evaluation results are shown at the end of each section. After presenting all of the results for each part, Section 6.4 completes the chapter with a discussion.

6.1 Part 1

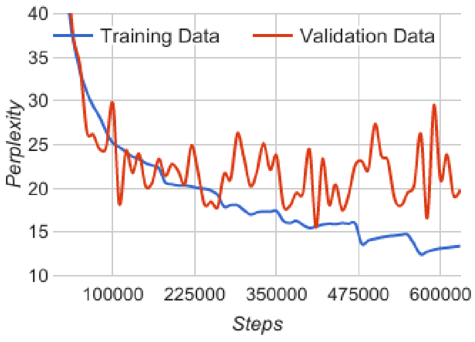
Part 1 is trained on the UDC dataset, focusing on single questions. This part includes three models; the Grid LSTM, LSTM, and GRU models.

6.1.1 Perplexity

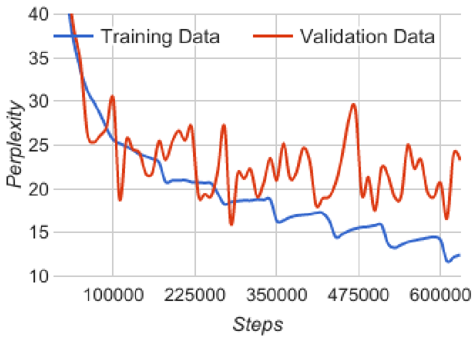
In order to monitor the progression of a model, every model performs two perplexity calculations during training. First, it calculates how well the model has predicted the training responses by looking at the average loss from the previous 10000 steps. Second, each model runs one step using the validation data, to calculate how well it predicts a sample it has never encountered before. Figure 6.1 shows the perplexity of the Grid LSTM, LSTM, and GRU models, both for the training and validation data. As expected, the training perplexity is decreasing continuously, while the validation set perplexity has a fluctuating, but slightly decreasing development. This fluctuation is discussed in Section 6.4. The training data perplexity forms a "staircase pattern", where the perplexity does a significant drop at the start of a new epoch. This phenomenon is a result of our batchwise training, where the



(a) Grid LSTM



(b) LSTM



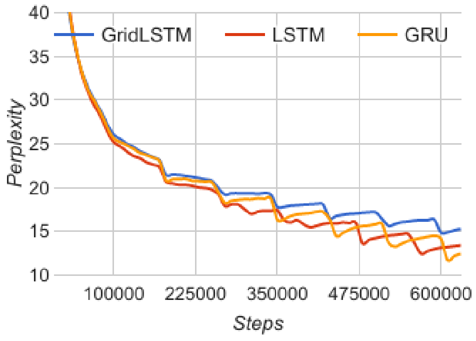
(c) GRU

Figure 6.1: The perplexity of the training and the validation set for the Grid LSTM, LSTM, and GRU model

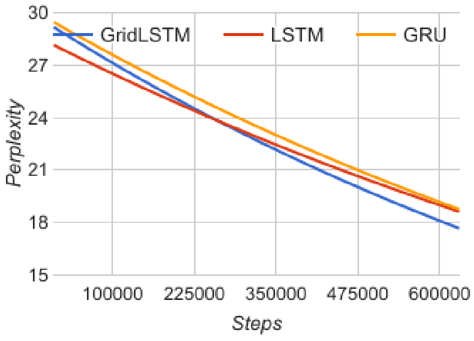
training pairs are trained exactly once every epoch. Consequently, we obtain a lower perplexity when the next epoch starts since the equivalent training pairs are revisited.

Figure 6.2a compares the training perplexity for the three models. The Grid LSTM model scores worse than the LSTM and GRU models, whereas the two latter follow a similar curve.

The validation set perplexity drawn in Figure 6.1 is difficult to compare, due to the fluctuation. Therefore, we obtained the trend lines from each model by doing an exponential regression of the curves. The result is shown in Figure 6.2b. It is interesting that the Grid LSTM model now receives the best perplexity score.



(a) UDC Training data perplexity



(b) UDC Validation data perplexity

Figure 6.2: A comparison of UDC training and validation data perplexity for GridLSTM, LSTM and GRU

6.1.2 Conversational Agents Outputs

The Human Evaluation for Part 1 consists of twelve questions. These questions and the corresponding results are listed in Table 6.1. Because we have created technical support chatbots, with specialization at Ubuntu problems, we provided the evaluators with additional information to make them better equipped to rate some of the responses.

Question 1	Hi, I'm new here. I have a question, can you help me?
Dataset:	What is the problem?
Grid LSTM:	Ask away. What is the Question?
LSTM:	Shoot. Just ask.
GRU:	Just ask. If someone knows they will answer.

Question 2	Hi there all. A friend just accidentally formatted the wrong ntfs drive. Are there any linux programs that can recover that sort of thing?
Additional info:	• *Using dd you can create backups of an entire harddisk or just a parts of it.*

	• *Chkdsk verifies the file system integrity of a volume and fixes logical file system errors. It is similar to the fsck command in Unix.* • *TestDisk is a free data recovery software*
Dataset:	Quick formatting, or full formatting?
Grid LSTM:	You can use dd. Or dd.
LSTM:	Chkdsk the drive. Or fsck.
GRU:	Testdisk. Or testdisk.

Question 3	How do i copy a file from a remote server to my local machine?
Additional info:	• *SCP (Secure Copy) is a command line tool for Linux systems for securely transfer files from local to remote server or vice a versa.* • *Remote to Local example: scp user@192.168.10.100:/remote/file.txt /path/to/local/*
Dataset:	Scp user@host:_DIR _DIR
Grid LSTM:	Scp.
LSTM:	Scp. Scp user@host:_DIR user@host:_DIR.
GRU:	Scp. Scp remotehost user@host:_DIR.

Question 4	How to close a hanged program?
Additional info:	• *Kill - Kill the process by specifying its PID* • *Killall - Kill processes by name* • *Xkill - kill a client by X resource*
Dataset:	Kill.
Grid LSTM:	Xkill.
LSTM:	Killall -9 <app>. Killall -9 programname.
GRU:	Killall <app>. Or xkill.

Question 5	Hi, when is the next ubuntu release due? The faq dont say
Additional info:	● *It was in april.*
Dataset:	March for preview, april for final.
Grid LSTM:	April 19th. _DIR.
LSTM:	April 29th. Yes.
GRU:	April. I think.

Question 6	Hello. What is the difference between xchat and xchat-gnome
Additional info:	● *Xchat-gnome is a different GUI for xchat*
Dataset:	Xchat-gnome is more of a stripped down version of the original xchat. No problem mate
Grid LSTM:	Xchat is the default.
LSTM:	Xchat-gnome is the gui. Xchat-gnome is the default.
GRU:	Xchat is xchat. Xchat-gnome is the name of the protocol.

Question 7	How do i ping in ubuntu?
Dataset:	Type ping your_site.com
Grid LSTM:	Ping _URL.
LSTM:	Ping _URL.
GRU:	With ping. Ping.

Question 8	How can i boot into text mode and run fsck?

Additional info:	● *GRUB is a boot loader package.* ● *Recovery mode just loads some basic services and drops you into command line mode.* ● *Text mode = Terminal/commando line* ● *Fcsk = File system consistency check*
Dataset:	At the grub menu when it first boots, choose recovery mode.
Grid LSTM:	Boot into rescue mode. And boot into recovery mode.
LSTM:	You can use the recovery console. Sudo apt-get install fsck.
GRU:	Yes. Boot into recovery mode.

Question 9	Is there a way I can assign any program I want to a shortcut key in gnome?
Additional info:	● *GNOME is a desktop environment composed of free and open-source software that runs on Linux and most BSD derivatives.* ● *Xbindkeys is a program that allows to bind commands to certain keys or key combinations on the keyboard.*
Dataset:	System - preferences - keyboard shortcuts.
Grid LSTM:	System -> preferences -> keyboard shortcuts. You can use the keyboard shortcuts in gnome.
LSTM:	Yes. System -> preferences -> keyboard shortcuts.
GRU:	Xbindkeys.

Question 10	Is it possible to build a driver on a livecd and use it? If I just build the module and use modprobe will it work?
Additional info:	● *A liveCD is a CD containing a bootable computer operating system.*

	● *Modprobe is used to add a loadable kernel module (LKM) to the Linux kernel.*
Dataset:	You can if you build it against the system kernel.
Grid LSTM:	Yes. You can use the module to build it.
LSTM:	Yes. What is the problem?
GRU:	No. You need to compile it.

Question 11	How to install debian package?
Additional info:	● *sudo were designed to run commands only as the superuser.* ● *apt-get is the command-line tool for handling packages (installing/removing)* ● *Aptitude is a tool for package management in Debian GNU/Linux*
Dataset:	Apt-get install <package name>. Or aptitude install <package name>.
Grid LSTM:	Sudo apt-get install packagename. Sudo apt-get install.
LSTM:	You have to install it. _URL.
GRU:	What package?

Question 12	How would security updates be handled for the operating system and application software if the user doesn't have root password?
Additional info:	● *Sudo passwd root tells the system to change the root password, and to do it as though you were root*
Dataset:	Becouse by default the first account is part of the sudoers admin group.
Grid LSTM:	_URL. You can use the command line to run a program.
LSTM:	You can use sudo. Sudo passwd root.
GRU:	You can set security updates. But security updates are not supported.

Table 6.1: Part 1

6.1.3 Human Evaluation

	Grammar	Content	Total score
Dataset	3.86	3.69	3.77
Grid LSTM	3.59	3.00	3.29
LSTM	3.61	3.01	3.31
GRU	3.45	2.91	3.18

Table 6.2: Part 1: Human Evaluation results concerning single questions from the UDC conversations.

Table 6.2 presents the results of the HE. The actual response from the dataset is, not surprisingly, superior to the chatbots. Nevertheless, it is interesting that the test set does not receive an even better score. The fact that the LSTM model obtains better results than the GRU model makes sense, considering that the GRU is a "simplification" of the LSTM cell. Also, the time reduction was not enough to compensate GRU's weaker results. The difference between the Grid LSTM and LSTM's total score is negligible and as small as 0.02. Hence, we can neither confirm nor refute that Grid LSTM cells will retrieve better results in the field of conversational agents.

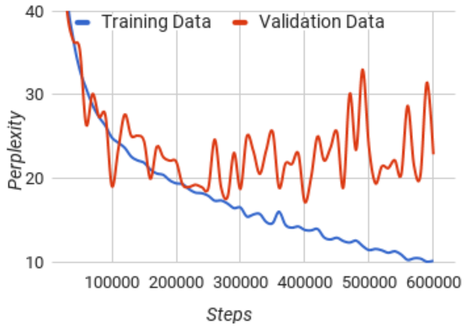
6.2 Part 2

In the second part of this experiment, the models are still within the closed-domain field of the UDC dataset, but where the models ability to handle shorter conversations is evaluated. The Context-Prepro model and the Stateful model is used in this part, in addition to the LSTM baseline.

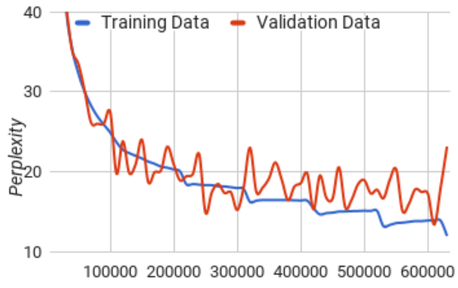
6.2.1 Perplexity

The perplexity of the different context approaches is depicted in Figure 6.3. Figure 6.4 compare the perplexities, where 6.4a looks at the training data, and 6.4b shows the perplexity of the validation data. Again, the results from the training data contradict the results from the validation data. The Stateful model receives the best training perplexity, whereas the Context-Prepro model scores better on the validation data. The reason for the more frequent drops for the Stateful model in

Figure 6.4a is the result of less training data and the fact that it is divided into two files instead of one.

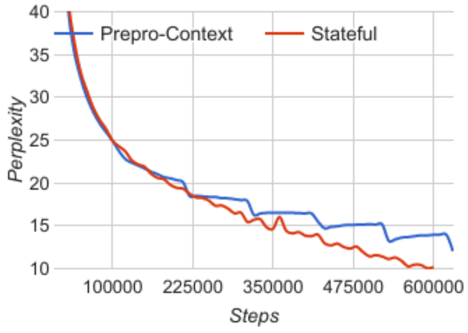


(a) Stateful Model

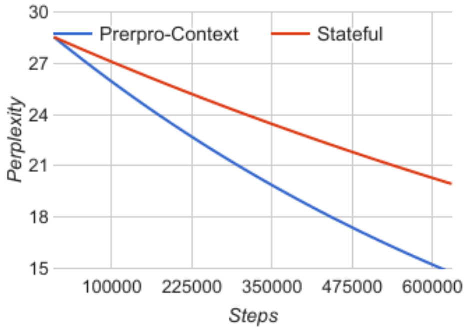


(b) Context-Prepro model

Figure 6.3: The perplexity of the training and the validation set for the Stateful and Context-Prepro model



(a) UDC Training data perplexity



(b) UDC Validation data perplexity

Figure 6.4: A comparison of UDC training and validation data perplexity for Context-Prepro and Stateful Model

6.2.2 Conversational Agents Outputs

This part of the experiment focuses on the ability for different conversational agents to hold a natural conversation on four different topics. All chatbots will be asked the same starting question, but the succeeding questions may differ as the conversation evolves.

	Conversation from Dataset
Human:	Is 64bit ubuntu faster than 32 bit?
Chatbot:	No
Human:	Why 64bit then?
Chatbot:	Ability to use more memory.

	Stateful Model Conversation
Human:	Is 64bit ubuntu faster than 32 bit?
Chatbot:	No. It is not.
Human:	Why not?
Chatbot:	Because it is a bad idea. I would not think it would be a good idea to use 64bit.
Human:	Can it use more memory?
Chatbot:	It can. But it is not recommended.

	LSTM Model converesation
Human:	Is 64bit ubuntu faster than 32 bit?
Chatbot:	No.
Human:	Why 64bit then?
Chatbot:	X86_64 is for 64bit processors. 32bit.
Human:	Why is 64 bit better than 32 bit?
Chatbot:	Why not? It is easier.

	Prepro-Context Model Conversation
Human:	Is 64bit ubuntu faster than 32 bit?
Chatbot:	Depends on your processor. :)
Human:	Intel core i7
Chatbot:	What processor?
Human:	Just told you
Chatbot:	What? What is that?
Human:	I think it is good enough for 64 bit
Chatbot:	I do not know. I do not use 64bit.

Table 6.3: Part 2 - Topic 1

	Conversation from Dataset
Human:	To anyone, does linux have a movie maker program like windows does?
Chatbot:	Kino.
Human:	What is it called?
Chatbot:	I just told you :)
Human:	Lol i thought that was a typo. What is the command to install kino?
Chatbot:	Sudo aptitude install kino.
Human:	Thanks.
Chatbot:	Of course. It is also installable via apt-get or synaptic. I just prefer aptitude...

	Stateful Model Conversation
Human:	To anyone, does linux have a movie maker program like windows does?
Chatbot:	Kino. !info kino.
Human:	How do i install it?
Chatbot:	_URL.
Human:	Can you tell me the command?
Chatbot:	sudo apt-get install cheese. sudo apt-get install vlc.

	LSTM Model Conversation
Human:	To anyone, does linux have a movie maker program like windows does?
Chatbot:	Try kino. _URL.
Human:	What is it called?
Chatbot:	Simple scan. _URL.
Human:	Is kino and simple scan the same?
Chatbot:	Yep. It is a gui app.
Human:	Wait, simple scan is a frontend for sane. Thats not what i want.
Chatbot:	Yes, i know. I am just saying.
Human:	What is the command to install kino?
Chatbot:	Yes. Sudo apt-get install kino
Human:	Thanks

Chatbot:	Np

	Prepro-Context Model Conversation
Human:	To anyone, does linux have a movie maker program like windows does?
Chatbot:	Kino. _URL.
Human:	How do i install it?
Chatbot:	Sudo apt-get install supertux.
Human:	That is not it. How do i install kino?
Chatbot:	Sudo apt-get install kino.
Human:	Lol. That is supertux?
Chatbot:	It is a program. It is a frontend.
Human:	So kino is a movie maker program?
Chatbot:	Yes. It is a program.

Table 6.4: Part 2 - Topic 2

	Conversation from Dataset
Human:	Can anyone help me to play video full screen in mplayer?? I cannot play realplayer format in full screen in mplayer. In fact after installing w32codec, the mplayer can not play anything in full screen. I do not like realplayer.
Chatbot:	you want to edit your _DIR.
Human:	How?
Chatbot:	With any text editor.
Human:	I mean what do i need to edit? Before i install the w32codec, I can double click mplayer to get to full screen.
Chatbot:	There will be lines that say: . # start in fullscreen mode by default. Skip_checks=yes.
Human:	But now i double click, It gets to full screen but the video is not to fullscreen.
Chatbot:	Remove the # from skip_checks=yes.
Human:	I have that # start in fullscreen mode by default. Skip_checks=yes.

Chatbot:	The # is a comment character, remove it and mplayer will get the option skip_checks=yes. So that should start videos in full screen for you.
Human:	I still cannot get the full screen working.
Chatbot:	Hm.
Human:	Do u have mplayer installed there?
Chatbot:	Yes.

	Stateful Model Conversation
Human:	Can anyone help me to play video full screen in mplayer??
Chatbot:	Install vlc. Sudo apt-get install vlc.
Human:	But i really want to use mplayer
Chatbot:	I do not know. I do not use totem.
Human:	You think vlc is better than mplayer?
Chatbot:	I think so. I do not know.
Human:	What is better?
Chatbot:	It is a matter of choice. i think it is better to use vlc.
Human:	Okay, and it will play full screen?
Chatbot:	I do not know. I do not use it .
Human:	Why did you recommend it then?
Chatbot:	I said it is better. I do not use it.
Human:	Yes, but you did not tell why
Chatbot:	I did not say anything I said i was not.

	LSTM Model Conversation
Human:	Can anyone help me to play video full screen in mplayer??
Chatbot:	What video card? What video chip?
Human:	Video card gtx 1080
Chatbot:	Yes it is . _URL
Human:	But how can i get full screen?
Chatbot:	Try f11. _URL.
Human:	Does not work for mplayer
Chatbot:	It does.

	Prepro-Context Model Conversation
Human:	Can anyone help me to play video full screen in mplayer??
Chatbot:	Install vlc. Or vlc.
Human:	But i really want to use mplayer
Chatbot:	Then install vlc. Or vlc.
Human:	You think vlc is better than mplayer?
Chatbot:	I use vlc. It is better.
Human:	What is better?
Chatbot:	Same problem. I use vlc.

Table 6.5: Part 2 - Topic 3

	Conversation from Dataset
Human:	What is the sudo root -i command so i can login as root. Anyone please?
Chatbot:	Su - root .
Human:	It does not work, asks for a password and rejects it.
Chatbot:	Sudo passwd.

	Stateful Model Conversation
Human:	What is the sudo root -i command so i can login as root. Anyone please?
Chatbot:	Sudo -i.
Human:	It does not work, asks for a password and rejects it.
Chatbot:	What is the error? What is the error you get?
Human:	Password incorrect
Chatbot:	No password. You can use sudo.
Human:	How?
Chatbot:	Sudo passwd username. Then sudo _DIR.

	LSTM Model Conversation
Human:	What is the sudo root –i command so i can login as root. Anyone please?
Chatbot:	Sudo –i . Sudo –s
Human:	It does not work, asks for a password and rejects it .
Chatbot:	What is the password? And what does it say?
Human:	Haha, i am not telling you my password. It rejects it
Chatbot:	:) ok
Human:	Do you know about another way to log in as root?
Chatbot:	Yes. Sudo passwd.
Human:	That did not work.
Chatbot:	What? It did not work?

	Prepro–Context Model Conversation
Human:	What is the sudo root –i command so i can login as root. Anyone please?
Chatbot:	Sudo –i.
Human:	it does not work, asks for a password and rejects it.
Chatbot:	Use sudo. Sudo –i.
Human:	I tried, but it did not work
Chatbot:	What did you do? Sudo –i?
Human:	Yes, and it said that the password was incorrect.
Chatbot:	And you did not do sudo –i? What is the problem?
Human:	I did yes
Chatbot:	What is the problem?

Table 6.6: Part 2 - Topic 4

	Grammar	**Content**	**Total score**
Dataset	4.23	3.98	4.1
Stateful	3.80	2.71	3.25
LSTM	3.78	2.38	3.08
Context-Prepro	3.75	2.08	2.92

Table 6.7: Part 2: Human Evaluation results concerning conversations from the UDC conversations

6.2.3 Human Evaluation

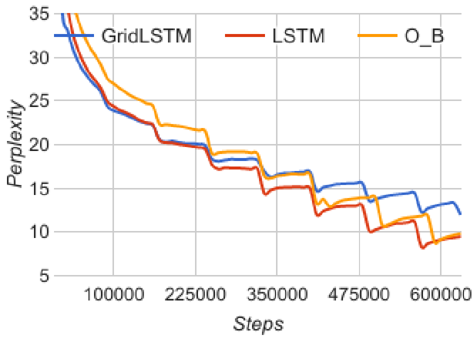
The agents tested in Part 2 score worse than the models in Part 1. However, this part focuses on a greater problem, as it does not only consider single questions, but rather an entire conversation. The Stateful model receives the highest score among the chatbots, while the baseline beats the Context-Prepro model.

6.3 Part 3

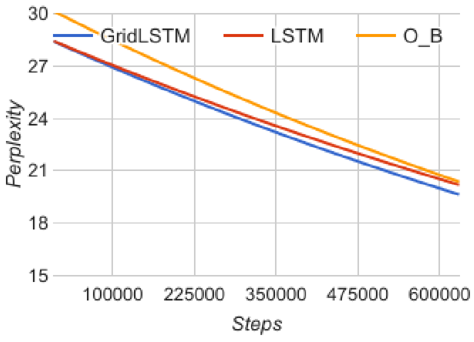
The final part of the experiment is there to increase the external validity to the research project. This time, however, the baseline, Grid LSTM, Stateful-Decoder and One-Bucket models are trained on the OpenSubtitles dataset.

6.3.1 Perplexity

The perplexity results in Part 3 are similar to the results in Part 1. Grid LSTM receives the worst score in Figure 6.5a, which looks at the training data, but obtains a better trend line in Figure 6.5b regarding the validation data.



(a) OS Training data perplexity



(b) OS Validation data perplexity

Figure 6.5: A comparison of OS training and validation data perplexity for Grid LSTM, LSTM and One-Bucket Model

6.3.2 Conversational Agents outputs

Part 3 uses the OpenSubtitles dataset to evaluate the models. Table 6.8-6.15 focus on single questions. Even though the questions is within the same topic, they do not necessarily share any context. The questions in table 6.16-6.21 are inspired from movie manuscript, and mimics fictive movie scenes.

1	`Hi`
2	`How are you?`
3	`What is your name?`
4	`Where are you from?`
5	`Where do you live now?`
6	`How old are you?`
7	`Are you a girl or a boy?`
8	`What do you study?`

Table 6.8: Part 3 - Introduction questions

	LSTM	**Grid LSTM**	**Stateful**	**One-Bucket**
1	Hello.	Hi.	Hi!	Hi!
2	Good.	Fine.	I am good.	I am good.
3	Kevin.	Sergei.	Franco.	Franco.
4	Guillermo.	Originally?	Australia.	Australia.
5	I am in the middle.	In a motel room.	I live in America.	I live in the hospital.
6	You are old.	Fifteen.	I am old enough.	I am old enough to meet.
7	I am a girl.	Boy.	Girl.	Girl.
8	I am a journalist.	Nothing.	I am a research.	I am a research.

Table 6.9: Part 3 - Introduction responses

1	Who are you?
2	What is your purpose?
3	What is your favorite food?
4	What is you favorite drink?
5	What is your favorite sport?
6	What is your favorite book?
7	Who is your boyfriend?
8	What is the purpose of your existence?

Table 6.10: Part 3 - Personal questions

	LSTM	**Grid LSTM**	**Stateful**	**One-Bucket**
1	I am the maid.	I am your friend.	I am the guy.	I am the guy.
2	It is my watch.	What is this?	I am a servant of god.	I am a servant of god
3	Food food.	Meat loaf.	Chicken.	Chicken.
4	I am a journalist.	Scotch.	Whiskey.	Whiskey.
5	What?	Blue.	Excuse me?	Football.
6	I do not know.	Fiction?	What?	What is this?
7	My boyfriend.	Jimmy.	My name is Jimmy.	Jimmy.
8	The memories of my life?	It is true.	Shut up!	You are a joke.

Table 6.11: Part 3 - Personal responses

1	Okay, so I am going to ask you some questions. Ready?
2	Is earth or mercury closer to the sun?
3	How many percent of the earth's surface is water?
4	What is a computer?
5	What does the fox say?
6	I want to loose weight, what should I do?
7	I want to gain weight, what should I do?
8	I am so hungry, what should I do?

Table 6.12: Part 3 - Knowledge questions

	LSTM	**Grid LSTM**	**Stateful**	**One-Bucket**
1	Sure.	Yeah.	I am not ready to go.	I am not ready to go.
2	Tomorrow.	Yeah.	I am not really sure.	I am not going to let.
3	About four.	No.	7.	7.
4	It is a bomb.	A diamond.	It is a.	It is a code.
5	The warden has been over.	It is not important.	That is the problem.	She is right.
6	You should not i know you are sick.	Nothing.	What should i do?	Report!
7	You should not be a bit more fun.	Whatever	What should i do?	I am a fucking asshole.
8	What should we do?	I am starving.	Eat it.	Well, i do not know.

Table 6.13: Part 3 - Knowledge responses

1	Can a bird fly?
2	Can a kengaroo fly?
3	Do you believe in love at first sight?
4	Is ten greater than one?
5	Is the grass yellow?
6	Is Darth Vader your father?
7	Do you know what time it is?
8	Is Trondheim in Norway?

Table 6.14: Part 3 - Yes/No questions

	LSTM	**Grid LSTM**	**Stateful**	**One-Bucket**
1	Bird fly?	Yeah.	How about a bowl?	Hey, hey, hey.
2	Yeah.	No.	You are supposed to be a pilot.	You are a very clever little.
3	I know.	Yes.	I believe in god.	I wish I knew, but.
4	I am a little surprised.	Yes.	That is not a good time.	I do not know.
5	Yeah.	Yeah.	No, it is a fireplace.	I am afraid it is.
6	Yes.	Yes.	I think he is the one.	My father?
7	I am late.	It is 8 : 30.	No.	It is a little bit more.
8	No way.	No.	Yes, my dad.	Do you know what that means.

Table 6.15: Part 3 - Yes/No responses

1	Do you know what day it is today?
2	It is ten years since our mother died.
3	I miss her so much. Do you miss her?
4	And we still do not know who killed her.
5	Do you have any ideas about who the killer might be?
6	Why are you smiling like that... What is going on?
7	Oh my god. What do you know about the murder?
8	Did you really love her?
9	I know you guys had some issues, but she was our mom. What do you know about the murder?
10	No way. You killed her, didn't you?

Table 6.16: Part 3 - Dramatic questions

	LSTM	**Grid LSTM**	**Stateful**	**One-Bucket**
1	The day after tomorrow.	Saturday.	February.	Today.
2	I do not blame her.	Now, listen	I am sorry.	My mother did not want to.
3	I guess so.	No	I really am her mother.	I do not know.
4	She is a ghost.	Who?	You are not a lawyer.	You are not supposed to know.
5	He did.	I do not know.	No, not really.	Come on, man.
6	I know.	I am sorry.	I have always known how to make a living.	I am not sure.
7	A murder?	Nothing.	They are not here.	I know the code.
8	I did.	No.	No.	Love?
9	And the end of the war was a little more complicated than that.	She was a fucking big shot.	That is a good point.	That is not what i meant.
10	She killed my wife.	You killed her.	I did.	What?

Table 6.17: Part 3 - Dramatic responses

1	I have been thinking a lot lately. And I have so many questions
2	What defines you? What makes you who you are?
3	Why do we exist in this universe? What is our purpose on earth?
4	I did not even choose to be a human being. Why are you a chatbot?
5	Why is there an universe? When and why was it created?
6	Who created the world? The big bang? God?
7	If god created the universe, who created god?
8	Today, I exist. Tomorrow I might be gone. Gone forever
9	What happens when we are dead?
10	Will we just be a part of the infinite dark? Will we become nothing? Just dissapear...

Table 6.18: Part 3 - Philosophical questions

	LSTM	**Grid LSTM**	**Stateful**	**One-Bucket**
1	I am not a professional.	Questions?	Good things have a good idea.	And?
2	You are not my girl.	You are a woman.	I do not know.	A lawyer?
3	I do not know.	We will find our path.	Our plans are not worth it.	What is this?
4	And a man.	To comfort me.	I am not a monster.	What is that?
5	You were right.	It was a miracle.	There is no question.	What is that?
6	A rocket?	God bless America.	You mean, you saved my life.	Now, what is this?
7	No one?	Who cares?	I do not know, but he is a god damn lunatic.	Do not know, I am not a.
8	Forever, forever!	No!	I am going to be a family.	I am not going to leave you.
9	I am not dead.	We die.	We are not dead!	We will take the next one
10	You are so fucking good.	No.	That is the only way we can get to the bottom of the earth.	You know, I am not really a criminal.

Table 6.19: Part 3 - Philosophical responses

1	Do you remember when we first met? On the beach. One year ago. You were beautiful...
2	I could not get your out of my head. Even though we were both seeing someone else...
3	But now, i meet you here... And we are both free. And you are just as beautiful as you were when i saw you last time.
4	Do you believe in love at first sight? Because I fell in love with you the second I saw you.
5	Do you feel the connection we have? The amazing chemistry?
6	It feels like fireworks, like Christmas, like home.
7	Just looking at you, makes me the happiest man on earth.
8	Please, marry me!
9	What do you mean? Don't you love me?
10	I will never give up. Never. You will be my wife. One day, you will say yes, right?

Table 6.20: Part 3 - Love questions

	LSTM	**Grid LSTM**	**Stateful**	**One-Bucket**
1	I remember that.	Yeah.	Remember that?	Remember that?
2	I was thinking about it.	Yes.	What?	We were like brothers.
3	Yes, I know.	And we are all adults.	You are fired.	I like you.
4	Yeah, yeah, yeah, baby.	I love you.	I will not let you.	She fell from the both of you!
5	Please.	Amazing.	I am really not.	Yes!
6	The end of the night.	Christmas eve.	It is a good spot.	It is like a dream.
7	God's earth.	I believe you would believe that.	A man in a red dress, a man like a man.	A single position.
8	No.	I am sorry.	I am not a monster anymore, I am not a killer anymore.	I do not know, i.
9	Love me.	I do not know.	I love you.	Or what?
10	Good.	I will.	Never.	Never.

Table 6.21: Part 3 - Love responses

6.3.3 Human Evaluation

	Grammar	**Content**	**Total score**
LSTM	3.91	2.67	3.29
Grid LSTM	4.14	3.26	3.70
Stateful-Decoder	3.97	2.67	3.32
One-Bucket	3.80	2.78	3.29

Table 6.22: Part 3: Human Evaluation results from OpenSubtitles

With an average score of 3.70 and a content score of 3.26, the Grid LSTM model is by far the superior model in this experiment. The LSTM baseline has an average score of 3.29, and these results indicates that the Grid LSTM outperforms the standard LSTM cell. Using a stateful decoder on a non-stateful model, did not yield any significant improvements to the results. In fact, when we look at the content score, the LSTM model trained on one bucket perform better than the equivalent model using the stateful decoder.

6.4 Discussion

This section discusses the results presented throughout this chapter.

6.4.1 Perplexity Results

We started each of the sections in this chapter by looking at the perplexity graphs of the models. The staircase pattern observed in all the perplexity curves obtained from the training data is an interesting topic to discuss. The reason for the sudden and significant drop is how we perform the batchwise training. Our approach guarantees that all of the training pairs in the training data is processed exactly once every epoch. Even though the model learns continuously during training, the revisit of a training pair will result in an even lower perplexity, visible as a drop in the graph. Other batchwise approaches, such as the one used in TensorFlow's translation model, will not lead to the same behavior. Their methodology is to load the entire training data into memory and randomly pick samples from it. This approach can neither guarantee that all of the training pairs are used during training nor tell how many times a particular pair is selected during an epoch. In extreme scenarios, the identical training data may be chosen by the batch every time, causing an overfitted and useless model. In the case of TensorFlow's batch generating approach, the graphs will not show a staircase pattern, but rather a more continuous graph without the drops. However, the argument of training all pairs the equal number of times should be superior compared to obtaining a continuous curve in the graph.

The perplexity sketched for the validation data is far more fluctuating than the curves observed for the training data. This fluctuation makes it hard to visualize the real performance of the model, which is why we used exponential regression to create smooth estimations of the curves. This regression could be done in various manners, but we believe that the exponential reflects the curves the best. There are several reasons for why the perplexity of the validation data does not follow a smooth curve. The first most intuitive case is that the batch size is too small. By using a bigger batch size, more validation data would be utilized in the calculation of the average perplexity, and thus reduce the problem. Another strategy to include more validation data for each perplexity calculation is simply by running multiple steps, with multiple batches of validation data and compute the average. A more drastic reason for the observed fluctuation may be that the validation set is not properly representing the rest of the dataset. However, we do shuffle the dataset before we divide it into training, validation and test data, with the exact purpose of diminishing this problem. A third reason may be that the models are not good enough, but this contradicts the fact that the models manage to lower the training perplexity scores.

By focusing on the results from the perplexity graphs from Part 1 (Figure 6.2), the Grid LSTM model obtains the worst score on the training data. However, the trend lines for the validation data show the opposite. Note that the perplexity of the validation data reflects the true performance of the model better than the training perplexity, as this is the measurement of how well the model can predict unseen data. It is therefore interesting that the Grid LSTM model also obtains the best score in Part 3, illustrated in Figure 6.5b.

In Part 2, where the focus is on entire conversations, the same contradiction between the two perplexity measurements happens once more. While the Stateful model obtains a better score on the training perplexity, the Context-Prepro model achieves a better trend line. However, in contrast to the other experiments, the prerequisites for Part 2 are a bit changed. Section 5.1.3 explains that the Stateful model has less training data, resulting in shorter epochs which are visible from the more frequent drops in Figure 6.4a. As the only model in this experiment, the validation data perplexity seems to have reached a minimum, indicating that the model may have converged. This is illustrated in Figure 6.3a. In this circumstance, the exponential regression trend line, might not be the most suitable estimation and might give a wrong impression of the model's behavior. Therefore, we do not consider the validation perplexity results as important as in the other parts.

6.4.2 Results from the Human Evaluations

All of the sections ended with a presentation of the results from the HE. Even though the test set received the highest score in Part 1 and 2, we find it interesting that it did not end up with a better score. This indicates that the training data still has its weaknesses, and suffer from noise. After preprocessing the UDC dataset, we obtain 2.3M turns. We could define stricter requirements, to avoid poor question-response pairs, and thus increase the quality of the training data. Another reason for the low score may be that our customized metrics are too strict. Participants of the evaluation may think that a reasonable response that makes sense does not deserve a rating that is described as "Response makes perfectly sense". By looking at the results in Part 1, it was difficult to conclude whether or not a model is superior to the other, due to the minimal score difference. However, we decided to discard the GRU model, as it obtained the lowest score and the training time did not make up for this.

The HE in Part 2, concerning the context-based approaches, the Stateful model achieved best results. The difference between this model's content score and the baseline's is 0.33, which indicates that the Stateful architecture manages to capture information from previous turns better than non-stateful models. The conversation with the Stateful model in Table 6.3 illustrates that passing the previous state to the next turn, helped the chatbot when interpreting the conversation. The chatbot's response reflected the context, even though the human question did not contain any information about the topic (whether to use 64-bit Ubuntu or 32-bit). The human types "Why not?" and the chatbot responds with "Because it is a bad idea. I would not think it would be a good idea to use 64bit". As in Part 1, the test set is superior again, and this time with a greater margin. The increased score for the test set makes sense as longer conversations with reasonable content are more impressive than single sentences. Another interesting observation is that the LSTM model in Part 1 has a score of 3.31, whereas the identical model used for Part 2 receives a score of 3.08. These observations indicate that it is more difficult for chatbots to respond properly several times in a row, and to substitute a human in a conversation with several turns.

Unlike Part 1, Part 3 shows a significant difference between the Grid LSTM model and the baseline. A content score of ~0.5 points better than the next best model shows that the use of Grid LSTM cells improves the results for the chit-chatting task. The minimal effect on the results with the Stateful-Decoder in Part 3, and the fact that the Stateful model was superior in Part 2 indicates that the stateful decoder itself has little effect on non-stateful models. Another observation is that the models with a single bucket tend to generate longer responses compared to both the LSTM and Grid LSTM model, which are trained on multiple. This is the case in Table 6.19. The output is pruned to fit the bucket where the question is located, which in our case is the same length as the question. This may be a limitation of the model, as short questions do not always imply short responses.

6.4.3 Limitations of the Experiments

The chatbots' intelligence is, of course, highly influenced by the content of the training data. The fact that it is hard to extract good question-response pairs from the majority of the OpenSubtitles dataset will make it more difficult to create a satisfying conversational agent for the chit-chatting task.

The results are based on the questions we extracted from the UDC dataset and created in the case of the OpenSubtitles chatbots. The fact that we found and created the questions is not optimal. Ideally, someone with knowledge about the Ubuntu operating system should choose suitable questions and conversations for Part 1 and 2. For Part 3, we could have asked a group of friends to chat with the chatbots, and they could choose the questions for this part of the evaluation. Another drawback of the execution of the HE is that we did not inform the participants to ignore the lack of capital letters. All words were lower case, which may have affected the grammar score for some responses. However, the current decoder used by all models will now output sentences starting with a capital letter, in addition to the singular form of the first person: "I". Thus, the "Conversational Agent Outputs" presented in this chapter differ slightly from the questions and responses used in the questionnaires.

The lack of correlation between the perplexity measurements and the result of the HE supports our intuition about the absence of a good automatic evaluation metric. We believe the results from our customized evaluation reflects the quality of the responses better than the perplexity results.

The translation model created by TensorFlow introduced the usage of buckets to train more efficient. A major difference between the translation problem and conversation generation is that while translations tend to have a similar length of the inputs and outputs, this is not necessarily the case for questions and responses in a conversation. A short question may lead to a long response. When dealing with the buckets, the decoder will place the question from the user in the smallest suitable bucket, and then restrict the response to have the same length. In the experiment in Part 3, we also provided two models with a single bucket, to allow the chatbot to generate long responses on short questions. The results show that these models also tend to reply with longer responses than the others.

Chapter 7

Conclusion

Section 7.1 presents our research's contributions, while Section 7.2 answers the research questions based on the results presented and discussed in Chapter 6. Further, Section 7.3 explains the challenges we encountered during the master thesis. Throughout this research period, we have gathered ideas of possible improvements that could lead to better results. These will be presented in Section 7.4.

7.1 Contributions

Based on our research goals and questions presented in Chapter 1, we can summarize our contributions in the following way:

Our main contribution is a Stateful model, which is a result of our research on how adjustments to the Encoder-Decoder model can improve the model's ability to remember information from previous turns. The conversations generated by the Stateful model show that it outperforms the LSTM baseline when it comes to the content. It can produce responses that reflect the topic of the conversation, even though the question did not contain any information about the context. The results from the human evaluation, a questionnaire conducted by 30 participants, indicate that the Stateful model's responses are preferred over the baseline's.

The second contribution is the comparison of three different conversational agent models, where the only difference is the cell architecture. We evaluate the three models by looking at the perplexity score and the results of the human evaluation. Based on the results we discarded the GRU model but continued with the evaluation of the Grid LSTM and LSTM models in a final questionnaire taken by 50 participants. The results indicate that the choice of RNN cell can affect the quality of the output and that the Grid LSTM cell increases the content quality of the responses in the chit-chatting task, i.e. casual conversations and small talk.

The third contribution concerns how we preprocess the data and is an approach for eliminating out-of-vocabulary (OOV) words from the dataset. The most common way to handle OOV words is to replace them with a special unknown token, i.e. "UNK". Our approach uses a word representation model to obtain word embedding vectors for all words in the dataset, both the OOV words and the vocabulary words. Further, all OOV words will be replaced with the most similar word in the vocabulary. This is done by calculating the cosine similarity between the OOV word's embedding vector and the vocabulary words' embedding vectors. The vocabulary word that results in the smallest angular difference will be the replacement of the OOV word.

To be able to develop conversational agents, we needed an overview of the state-of-the-art architectures applicable for this task. Our research's fourth contribution is, therefore, an architectural overview of the state-of-the-art models which can be applied when creating conversational agents.

7.2 Answering the Research Questions

During this master's thesis, we have reached our two goals. First, we studied and compared the effect of different RNN cells. Second, we experimented with context-based approaches, to incorporate context management to the conversational agents. As a result, we have created nine distinct conversational agents. For the UDC dataset, we trained the LSTM baseline, GRU, Grid LSTM, Stateful and Context-Prepro model. The OpenSubtitles chatbots include the LSTM baseline, Grid LSTM, One-Bucket and the Stateful-Decoder model. These models have been trained and tested differently to collect results used to guide us in answering the research questions. We have worked with two different datasets; the UDC and the OpenSubtitles dataset. The UDC leads to technical support chatbots, whereas the models trained on the OpenSubtitles dataset are more suited for the chit-chatting task. Using the results presented in Chapter 6, we will answer the research questions.

Research question 1 *Will the use of different RNN cells in an Encoder-Decoder model have significantly effect on the quality of the outputs?*

The results from Part 1 and 3 contradict each other when it comes to answering the first research question. The difference between the chatbots' scores is minimal in Part 1, which indicates that the different RNN cells do not have significantly impact on the quality of the outputs. On the other hand, the Grid LSTM model achieves significantly better results in Part 3, where we test the models' ability to handle small talk and more "movie related" conversations. Due to the vast difference between their score in the final part, we believe the choice of RNN cell can affect the quality of the response.

Research question 2 *Will a conversational agent that keeps track of previous questions and responses, catch the context of a conversation better?*

The Stateful model yielded significantly better content score than the baseline. This indicates that the answer to the second research question is yes; a conversational agent that handle the information from previous questions and responses will catch the context of a conversation better.

The results are promising and indicate that the field of conversational agents can be further improved. However, more experiments should be conducted to verify these results, based on the drawbacks of our execution of the HE as described in Section 6.4.

7.3 Challenges

One of the challenges during this research project has been the implementation of the Grid LSTM cell. When we started this project, we discovered the Grid LSTM cell in TensorFlow's Contrib module. However, this implementation required certain changes in order to match the architecture described in Kalchbrenner et al. [2015]. Due to the lack of documentation, we tried to get confirmation on our implementation on forums such as Google Discussion[1] and StackOverflow[2]. We did get some feedback, but we are still not completely sure if we did this implementation correct according to the paper.

7.4 Future Work

This section presents possible improvements to the preprocessing of the datasets and to the decoder. Then, we present future research directions that might improve the state-of-the-art for conversational agents.

7.4.1 Exploring Methods for Finding the Nearest Neighbour in a High Dimensional Space

In our attempt to remove all OOV words, we calculate the cosine similarity between the embedded vectors of the OVV words and the vocabulary words. This is just one

[1]https://groups.google.com/a/tensorflow.org/forum/#!forum/discuss
[2]https://stackoverflow.com/

approach of finding the nearest neighbor in a high dimensional embedded space. The disadvantage of this method is that it only looks at the cosine of the angle between the vectors, which is just a judgment of orientation and not magnitude. Ideally, the vocabulary word located closest to the OOV word in the n-dimensional space should replace this OOV word. Therefore, other methods should be explored as substitutions for the cosine similarity, to be used in the process of eliminating OOV words.

7.4.2 Beam Search in a Decoder

The number of possible sentences to output is m^n, where m is the vocabulary size, and n is the sentence length. Traversing all of these possibilities is not just unwise, as this involves a lot of strange sentence combinations, but also impossible due to memory and time complexity. In the attempt of finding the sequence of words that together receive the highest probability score, we need a heuristic search algorithm. The easiest one, but far from the best, is the greedy search algorithm, which is currently used by our models. The greedy algorithm will choose the best option in the current stage, without reasoning about the future outcomes. This may result in a weak local optimum instead of the global one, as there is no guarantee that the best overall sentence is the one that starts with the first most predicted word. However, the greedy approach will only traverse one path, which makes it a very time efficient search. Another probably better alternative is the beam search algorithm. Beam search will investigate different predictions, and avoid devoting all its trust in the most attempting one. The beam size b decides the range of directions the algorithm will explore at every stage. The algorithm can, therefore, return the best sequence from b^n paths, resulting in a solution closer to the global optima. However, beam search can neither guarantee that the best solution is among these combinations.

7.4.3 Stateful Encoder-Decoder Architecture using Grid LSTM cells

Both the Grid LSTM and the Stateful model presented good results, compared to our baseline. The Stateful model uses LSTM cells, and it would, therefore, be interesting to see whether the use of Grid LSTM cells could gain even better results.

7.4.4 Hybrid Conversational Agent

An open domain conversational agent should ideally be able to converse about general topics and have some general knowledge. The conversational agents evaluated in Part 3, did not score very well on the “Knowledge Questions”. Therefore, one strategy to make a better open domain chatbot is to combine several models. A "Hybrid Conversational Agent" will consist of multiple models, which should be trained on different datasets. A model trained on a specific dataset should respond well to similar questions. Further, we need some classification model to classify the incoming question, and decide which model that is best suited to answer that question. The challenge in this task is to define the different domains each chatbot should cover, and extract good question-response pairs for the different models.

7.4.5 Hybrid Corpora

Another approach to improve the quality of an open domain conversational agent is to combine different datasets for the training data. There exist several smaller datasets that together could form a qualitative corpus for a conversational agent. Merging knowledge datasets such as questions from television shows like "Jeopardy"[3] and "Who Wants To Be A Millionaire?"[4] with different conversation dataset like "Cornell Movie-Dialogs"[5], "Movie-DiC"[6] and "Microsoft Research Social Media Conversation Corpus"[7], could result in a both qualitative and quantitative dataset.

[3]https://www.reddit.com/r/datasets/comments/1uyd0t/200000_jeopardy_questions_in_a_json_file/
[4]http://onlinelibrary.wiley.com/doi/10.1002/cpe.4168/full
[5]https://people.mpi-sws.org/ cristian/Cornell_Movie-Dialogs_Corpus.html
[6]http://www.aclweb.org/anthology/P12-2040
[7]https://www.microsoft.com/en-us/download/details.aspx?id=52375

Bibliography

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

Antoine Bordes, Jason Weston, and Nicolas Usunier. Open question answering with weakly supervised embedding models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 165–180. Springer, 2014.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Silje Christensen and Simen Johnsrud. Project-idi-rnnchatbot. `https://github.com/siljec/project-idi-rnnchatbot`, 2017.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

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

Alessandro Duranti and Charles Goodwin. *Rethinking context: Language as an interactive phenomenon.* Number 11. Cambridge University Press, 1992.

Anthony Fader, Luke S Zettlemoyer, and Oren Etzioni. Paraphrase-driven learning for open question answering. In *ACL (1)*, pages 1608–1618. Citeseer, 2013.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016. URL `http://www.deeplearningbook.org`.

A. Graves, S. Fernandez, and J. Schmidhuber. Multi-Dimensional Recurrent Neural Networks. *ArXiv e-prints*, May 2007.

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

Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, volume 5. GMD-Forschungszentrum Informationstechnik, 2002.

Zongcheng Ji, Zhengdong Lu, and Hang Li. An information retrieval approach to short text conversation. *arXiv preprint arXiv:1408.6988*, 2014.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid long short-term memory. *arXiv preprint arXiv:1507.01526*, 2015.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ryan Lowe, Nissan Pow, Iulian Serban, and Joelle Pineau. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. *arXiv preprint arXiv:1506.08909*, 2015.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013b.

George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

Briony J Oates. *Researching information systems and computing*. Sage, 2005.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

Alan Ritter, Colin Cherry, and William B Dolan. Data-driven response generation in social media. In *Proceedings of the conference on empirical methods in natural language processing*, pages 583–593. Association for Computational Linguistics, 2011.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

Lifeng Shang, Zhengdong Lu, and Hang Li. Neural responding machine for short-text conversation. *arXiv preprint arXiv:1503.02364*, 2015.

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. A neural network approach to context-sensitive generation of conversational responses. *arXiv preprint arXiv:1506.06714*, 2015.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

Appendix A

UDC Content Examples

The following conversations are from the Ubuntu Dialogue Corpus. We have tried to find dialogues which reflect the characteristics of the dataset, and have not used any preprocessing on the following data. Table A.1, A.2, and A.3 show technical conversations, whereas Table A.4 is an example of how a monologue might look like.

Jackrabbit:	can anyone help me?
ajavid:	ask
Jackrabbit:	its a program that gets ati video drivers becuase right now my screen is max lol
ajavid:	lspci|grep VGA
ajavid:	try to use packaged software in ubuntu as much as possible
Jackrabbit:	ok so what do I do with that command do I put it in terminal?
ajavid:	r5xx and below on ati == free 3d accel with xorg 7.4 and latest mesa in jaunty
ajavid:	for r6xx + only do you require the fglrx driver
ajavid:	whate exactly are you trying to do?
ajavid:	don't do random things like that
ajavid:	in console terminal, sudo aptitude pciutils; lspci|grep VGA
Jackrabbit:	ok is that all on line like: sudo aptitude pciutils; lspci|grep VGA
ajavid:	yes, ; is a bash newcommand delimiter
Jackrabbit:	ahhhh man I tell ya once my screen is fixed I cant wait to read and learn about this OS becuase I played with

Table A.1: UDC content 1

dutch	hey so is there anyone here who can point me to some information on getting nvidia drivers to work in jaunty?
dutch	im having interesting problems with it
mobi-sheep	SLi ?
dutch	you have any information on that?
mobi-sheep	http://www.darraghverschoyle.com/2009/03/enabling-sli-on-ubuntu-810/
mobi-sheep	Run "man nvidia-xconfig" and you'll find out what this does. :)

Table A.2: UDC content 2

erUSUL	try "sudo dpkg --configure -a" ?
erUSUL	do you have a DN server in your lan?
erUSUL	do you have a DNS server in your lan? doing the name resolution
blz	not that I know of
blz	i though hostnames were usually resolved automagically on the local net... like through the router or something
erUSUL	so how do you expect that the hostnames to be resolved? you can use avahi/zeroconf automatic hostname resolution. hostname is hostname.local
erUSUL	so try ≪ ping hostname.local ≫
blz	ping hostname.local yields "ping: unkonwn hostname.local"
erUSUL	do it with an actual hostname instead of the string "hostname" i guess your machines have another names
blz	oh... derp
blz	ok that works
erUSUL	no problem
blz	i'm getting normal looking ping
blz	so the local machine's hostname works when I ping it... what's the next step?
erUSUL	it already works. you got your name resolution
blz	so now it should resolve?
erUSUL	you have to append .local to the machines hostnames
erUSUL	yes
blz	ok, but it doesn't
erUSUL	what program is failing ?
blz	to be clear, i ran the ping hostname.local on the local machine having the problem
erUSUL	ping works so it has to be a specific to that program problem
erUSUL	maybe in windows you have to enable zeroconf in some place ? ask in ##windows ?
blz	hmm maybe so... i'll check under a remote ubuntu system...

Table A.3: UDC content 3

sstoveld:	hey guys, im looking to install ubuntu for the first time. im a newb here, does it matter if i burn the iso to a DVD instead of a CD? im all out of CD’s
Svenstaro:	no wont matter
Svenstaro:	are you curretnly using windows?
Svenstaro:	lol, what a coincidence that you now want to try ubuntu
Svenstaro:	are you a gamer?
Svenstaro:	et me check on linux compability, but quite honsetly, hellgate sucks :<
Svenstaro:	you are better off playing diablo, hellgate doesnt run on linux as far as I can judge looking at wineapps db and diablo 2 runs like a dream out of the box, and it OWNS
Svenstaro:	no windows software runs natively on linux, it need to either be emulated or otherwise be made compatible, not all windows apps can be made to work on linux
Svenstaro:	...

Table A.4: UDC content 4

Appendix B

OpenSubtitles Content Examples

James Bond: For Your Eyes Only, 1981
good afternoon , mr. bond .
don’ t concern yourself with the pilot .
one of my less useful people .
you are now flying remote control airways .
think twice , 007 .
it’ s a long way down .
i’ ve looked forward to this moment , mr. bond .
i intend to enjoy it to the full .
really , have you no respect for the dead ?
good bye , mr. bond .
i trust you had a pleasant ... fright .
you are fading from my picture , mr. bond .
but the end cannot be far away .
mr. bond !
Virgin Stripped Bare by Her Bachelors (Oh! Soo-jung), 2000
<the vlrgln strlpped bare by her bachelors>
<day’ s walt ... >
you’ il have to wait outside .
will it be long ?
i’ m almost done .
okay .

```
hello ?
it’ s me .
soojung .
where are you ?
i just got up .
you mean just now ?
i’ m not feeling very well .
where are you ?
i’ m already here .
are you sick ?
i’ m sorry .
can’ t we do this another time ?
soojung ...
soojung , you know this wasn’ t easy .
if you’ re not really sick , please come .
do we really have to do it today ?
of course .
soojung , you’ re making things really difficult .
where are you ?
in the hotel ?
```

Gladiator, 2000

```
hail , mighty caesar !
caesar !
caesar !
caesar !
caesar !
we who are about to die salute you !
on this day ...  we reach back to hallowed antiquity ...
to bring you a re creation ...  of the second fall of
mighty carthage !
on the barren plain of zama ...  there stood the
invincible armies ...  of the barbarian hannibal .
ferocious mercenaries and warriors ...  from all brute
nations ...  bent on merciless ...  destruction ...
conquest .
your emperor ...  is pleased to give you ...  the
barbarian horde !
anyone here been in the army ?
yes .
i served with you at vindobona .
you can help me .
```

whatever comes out of these gates ... we’ ve got a
better chance of survival if we work together .
do you understand ?
if we stay together , we survive .
the emperor is pleased to bring you the legionnaires ...
of scipio africanus !
to the death !
kill !
kill !
kill !
stay close !
come together !
staggered columns !
staggered columns !

Intimate Stories, 2002

$< i >$hello ?
hello ? $< /i >$
$< i >$who is it ? $< /i >$
$< i >$hello ? $< /i >$
it seems he’ s not here .
he must be sleeping .
come on , let’ s wake him up , lazy bastard !
come on .
uglyface .
uglyface .
it’ s me .
it’ s me , uglyface .
uglyface .
uglyface .
come , uglyface , come .
uglyface .
uglyface .
here , here .
uglyface !
how are you ?
how are you ?

Fun with Dick and Jane, 2005

hey , veronica .
hi , how are you ?
good , good .
i didn’ t know you worked out here .

```
welcome to kostmart .
i hope you’ il take a trip by the deli today ...  ...
for a complimentary cube of jalapeno cheddar .
i’ m lactose intolerant .
where do you keep the cigarettes ?
behind the counter .
but i’ m not sure that’ s a good ...
dick ?
you missed one .
```

Table B.1: OpenSubtitles conversation examples