DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2017

End-to-End Trainable Chatbot for Restaurant Recommendations

AMANDA STRIGÉR

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

End-to-End Trainable Chatbot for Restaurant Recommendations

AMANDA STRIGÉR

Master in Computer Science
Date: July 6, 2017
Supervisor: Joel Brynielsson
Examiner: Olov Engwall
Swedish title: Neuronnätsbaserad chatbot för restaurangrekommendationer
School of Computer Science and Communication

Abstract

Task-oriented chatbots can be used to automate a specific task, such as finding a restaurant and making a reservation. Implementing such a conversational system can be difficult, requiring domain knowledge and handcrafted rules. The focus of this thesis was to evaluate the possibility of using a neural network-based model to create an end-to-end trainable chatbot that can automate a restaurant reservation service. For this purpose, a sequence-to-sequence model was implemented and trained on dialog data. The strengths and limitations of the system were evaluated and the prediction accuracy of the system was compared against several baselines. With our relatively simple model, we were able to achieve results comparable to the most advanced baseline model. The evaluation has shown some promising strengths of the system but also significant flaws that cannot be overlooked. The current model cannot be used as a standalone system to successfully conduct full conversations with the goal of making a restaurant reservation. The review has, however, contributed with a thorough examination of the current system, and shown where future work ought to be focused.

Sammanfattning

Chatbotar kan användas för att automatisera enkla uppgifter, som att hitta en restaurang och boka ett bord. Att skapa ett sådant konversationssystem kan dock vara svårt, tidskrävande, och kräva mycket domänkunskap. I denna uppsats undersöks om det är möjligt att använda ett neuralt nätverk för att skapa en chatbot som kan lära sig att automatisera en tjänst som hjälper användaren hitta en restaurang och boka ett bord. För att undersöka detta implementerades en så kallad "sequence-to-sequence"-modell som sedan tränades på domänspecifik dialogdata. Systemets styrkor och svagheter utvärderades och dess förmåga att generera korrekta svar jämfördes med flera andra modeller. Vår relativt enkla modell uppnådde liknande resultat som den mest avancerade av de andra modellerna. Resultaten visar modellens styrkor, men påvisar även signifikanta brister. Dessa brister gör att systemet, i sig självt, inte kan användas för att skapa en chatbot som kan hjälpa en användare att hitta en passande restaurang. Utvärderingen har dock bidragit med en grundlig undersökning av vilka fel som görs, vilket kan underlätta framtida arbete inom området.

Contents

Chapter 1

Introduction

Chatbots have a wide variety of possible use cases and are of interest to both the industry and researchers. More and more companies are considering adding bots to their messaging services. Chatbots are useful as they can answer questions and assist customers without delay and at all hours. The question is whether chatbot design techniques have become advanced enough for us to create bots that are useful in practice.

Chatbots of varying degree of intelligence have existed for years, the first appearing in the 60's [31]. These first ones used simple keyword matching to generate an answer. Since then much has happened and many more chatbots have been developed using different techniques such as pattern matching, keyword extraction, natural language processing, machine learning, and deep learning.

There are generally two different types of chatbots in regards to the intended use case, conversational bots and task-oriented bots. Conversational bots aim to entertain the user by simulating a general conversation, usually without a specific end goal. In this case, longer conversations are often an indication of a bot performing well. The aim of a task-oriented bot is, on the other hand, to interpret and perform the action the user is requesting as fast as possible.

Chatbots have many possible fields of application and especially task-oriented chatbots have a clear utility in real applications as they are built to help users achieve some specific goal such as to make a restaurant reservation, order a pizza, book a flight, and so on. Using bots instead of human assistants could reduce the time and effort required to get help with the specific task at hand.

There exist different ways of creating a chatbot. This thesis will focus on end-to-end trainable systems, which means that the entire model can be trained directly on conversational data without any previous domain knowledge. This approach has several advantages as it does not require feature engineering and can thus be used across domains as the model does not use any assumptions about the domain in the conversations.

1.1 Purpose and Research Question

The aim of this project is to look into the current state of chatbot design techniques and to evaluate the possibility of using end-to-end trainable conversational systems in task-oriented environments.

For this purpose, a chatbot was developed using sequence-to-sequence learning. The model learns a mapping between sentences, i.e., between the context and the appropriate response. The chatbot was trained to assist users in finding a restaurant and make a reservation. The bot had to learn directly from dialog data how to achieve the task without relying on feature engineering or handcrafted rules. The bot's ability to perform this task was evaluated as well as the strengths and limitations of the system.

In this work, the data used for training and evaluation consists of conversations where the goal is to make restaurant reservations (further described in Section 5.2). However, the results can be considered relevant for other domains as well as many task-oriented dialog systems have to handle the same type of tasks regardless of domain, such as understanding user requests, issuing API calls, and asking for complementary information. The information in itself, the results of API calls and how the bot is supposed to act on them might vary, but the basic structure could be similar in many areas.

The question to be answered in this thesis is: to what extent can a restaurant reservation system be automated using an end-to-end trained chatbot?

Chapter 2

Background

This chapter provides an introduction to chatbots. In the first section, two different types of chatbots are presented and advantages and disadvantages of both are discussed. Then follows a discussion of available datasets to use when training a learning system and different ways to evaluate a conversation system. The chapter is concluded with a discussion regarding common difficulties when designing a chatbot.

2.1 Chatbot Models

Apart from the previously mentioned division of chatbots into the categories of conversational and task-oriented bots based on the intended use case, there are further distinctions. It is common to talk about retrieval based models [10, 3] as compared to generative models [24, 25, 29]. Retrieval based models process the user input in some way and pick an answer from an existing set. The downside with this type of system is that we have to anticipate all possible use cases and construct a set of all possible answers, and are then limited to this set. This can make it hard to handle new or unexpected situations which makes this model unsuitable to use in an open domain. A positive aspect is that we can control what is in this set of possible responses and can guarantee the quality of the responses [10]. A generative model generates new sentences, which have possibly never been seen before, based on the user input [29]. Therefore, we do not have to design all possible answers beforehand [29]. The idea is that given a large amount of training data of human conversations, the bot should learn to model the language and be able to generate an answer. A problem with this model is that the answers often tend to be generic, irrelevant, inconsistent or grammatically wrong [15, 29].

2.2 Datasets

The availability of datasets to train the conversation system on varies depending on what type of data you need. For general conversational systems, different forums are often used [18, 23]. Publicly available task-oriented data is however scarce. Task-oriented data is often collected by companies or collected using Wizard of Oz [7] techniques and crowdsourcing [32]. Using this last-mentioned technique, data is obtained by letting users chat with what they believe to be an autonomous system which in reality is controlled by a human. An alternative to data collected from the real world is to use synthetically generated data such as the dataset created by Bordes et al. [3]. There are advantages and disadvantages with using both real and synthetic data. With synthetic data, you have more control over the test setting and what you are testing but it is often not representative for a real use case. Real data provides an accurate setting for the usage of the conversational system but contains different difficulties such as grammar mistakes, slang, abbreviations, and other human errors. While we want the system to be able to handle these conversations they can make the training more difficult. One way to improve the performance is to do some preprocessing of the raw data. A few examples of what can be done is to convert contractions to full words, do spelling and grammar correction, convert all letters to lower case, remove punctuation, and replace words such as names or numbers with a common token, e.g., <person>, <number>, and so on [24].

2.3 Evaluation

When developing a conversation system, an important question is how to evaluate its performance. Different systems have different properties and might need to focus on different aspects in the evaluation. This could be user entertainment or satisfaction, goal achievement, grammar, consistency, and so on.

One way to evaluate a conversation system is to use humans to judge the quality of the system. While possibly being the most accurate evaluation metric, it is impractical and subjective. The results are often hard to translate to a quantitative result which makes it difficult to compare different systems. Therefore, automatic measurements that are easy to produce, reproduce, and use for comparing systems, are needed.

The paper by Liu et al. [17] discusses a few evaluation metrics for dialog systems and how they correlate to human judges. The evaluation includes word-overlap based metrics, such as BLEU [22], that measure word-overlap between a ground-truth response and the response proposed by the system. Another alternative is embedding-based metrics. This metric is based on the idea that the meaning of a word is defined by its word embedding [17]. Embeddings are further discussed in Section 4.5. A sentence-level embedding is calculated and the embeddings of the response and the ground-truth are compared, for example using cosine distance [17]. Liu et al. [17] however found that none of the evaluated metrics corresponded sufficiently to human judges.

Retrieval based systems are generally evaluated based on if the retrieved answer is the same as the ground truth answer [17]. When the answer can be classified as either correct or incorrect, we can use standard metrics such as precision, recall, and accuracy [5, Ch. 11]. These metrics are more straightforward than the different similarity metrics. An answer is either correct or not. One problem is that it can be too strict. In many cases, multiple answers can be semantically equivalent and thus equally correct. One evaluation metric that takes this into account is recall@k. Recall@k measures how often the ground-truth response is among the k most likely responses [19].

Automatic evaluation is needed for easy performance evaluation and fast testing during development. Qualitative results can, however, be just as important. Human judges are the only accurate mean of evaluation for assessing qualitative aspects.

2.4 Difficulties

When creating a chatbot there are several issues and difficulties that need to be addressed. One difficulty is to keep track of user intent, i.e., what the user wants [10]. Since we are using natural language to communicate, there is a multitude of ways the user can express an intent and the system has to understand them all. If the system asks "do you have any preferences on a type of cuisine?" the user can answer in a myriad of ways which all have the same meaning. In addition, the user intent can depend on multiple variables and change over the course of the conversation.

Utterances are often dependent on the context in which they were given [26]. Both the context of what has been said previously in the conversation and the physical context can be relevant. The physical context is for example location, time, and information about the user. Keeping track of this context can be challenging as it grows. With a large context, it becomes more difficult to identify what parts of the context are relevant to the current question.

Another issue is the tendency of systems trained on real data to give common or generic responses [15]. Generic responses can be used in most situations but do not contribute to the conversation in any meaningful way. The system favors these safe options as the objective during training is to maximize the probability of success. Common expressions such as "yes", "no", and "thanks" are generally overrepresented in conversational data. This causes the system to see them as more likely and therefore a safe bet to use as an answer to almost anything. These problems can be combated by using different normalization techniques and anti-language models that are designed to penalize high-frequency and generic responses [15].

Creating a system that can respond in a consistent way or have a personality is difficult for learning systems [16]. This is because conversational systems are often trained on dialog data collected from many different users. As a result, the personality of the chatbot becomes a combination of all of them. The same question formulated in different ways can, therefore, receive very different responses. For example, the question "where were you born" can generate the answer "Stockholm" while the question "where do you come from" can receive the answer "Madrid". Personality and consistency are important if a chatbot is to seem human. Consistency is necessary for task-oriented chatbots since the user's goal should be achieved regardless of how it is formulated.

Chatbots, especially task-oriented bots, often need to be able to interact with a knowledge base to be able to use information about the world. There exist different ways to manage this integration but it is extra challenging for end-to-end trainable systems. The interaction with the knowledge base often breaks the differentiability of the system which can then no longer be trained purely end-to-end. Solutions include having several end-to-end trainable modules and training them separately [32] or to augment the conversations with the calls to the knowledge base [3].

2.5 Ethics

Chatbots can be used for many purposes and not all of them are good. Most people have probably heard of or encountered spambots. Spambots are programs that are designed to automatically send out unsolicited messages through different channels. The goal is usually to advertise, boost a site's search engine ranking, or to trick users in some way. As chatbots become more advanced they can become difficult to detect and thus more successful at scamming people.

Another issue that often arises when discussing artificial intelligence is the possibility of computers replacing humans by automating their work. This discussion very much applies to task-oriented chatbots as their entire purpose is to automate tasks. Automation does not, however, have to be a bad thing. If chatbots can automate simple and repetitive tasks, it can free up time for the human workers to focus on the more advanced cases. Workers will have to deal with less boring and repetitive tasks and users can get quicker responses and easier access to services.

Chapter 3

Related Work

In this chapter, a few conversation systems are presented and the different models used in the works are briefly discussed. First conversation systems used in task-oriented domains are presented. Following, different end-to-end trainable systems are introduced and finally, two end-to-end trainable systems for task-oriented conversations are discussed.

3.1 Task-Oriented Conversation Systems

One way to model conversations is as partially observable Markov decision processes (POMDPs) which have been used in spoken dialog systems [34, 36]. POMDPs are statistical systems where the user input is regarded as a noisy observation of the conversation state [34]. What action to take is determined based on this state. The aim of the model is to improve this action selection policy via a reward driven process [34]. One limitation of this model is that the state and action space must often be designed by hand which limits its usefulness.

Another common model is slot-filling [14, 30] where a set of information slots are predefined. The system then attempts to fill in these variables from the dialog. These systems are only usable in limited domains where the possible features the user might refer to are predictable. In an open domain, it would be impossible to predict all possible variables.

Similarly, rule-based systems rely on hand-crafted rules that can depend on word or pattern matches, dialog state, entity detection, and so on. The rules specify the action to be taken in each case. Creating adequate rules can be difficult even in narrow areas and will soon become impossible in broader domains.

These types of models require specific domain knowledge and often massive amounts of manual design to work well. While capable of giving good results in limited domains these models are not usable across domains and have difficulties handling broad domains.

3.2 End-to-End Trainable Systems

The goal of end-to-end trainable systems is to limit the amount of manual labor needed to create systems that are general, scalable, and capable of operating in broad domains. This type of system works by learning a mapping from dialog context to system responses directly from the training data. Early work on end-to-end systems includes Ritter et al. [23]. Their work is based on phrase-based Statistical Machine Translation and generates responses to Twitter status posts by translating from the context of a conversation to a response. Sordoni et al. [26] built further on this work and used a recurrent neural network (RNN) architecture that can use information from previous utterances in the conversation. The RNN encoder-decoder model from Cho et al. [4] has a similar structure. RNNs are commonly used for language modeling. RNNs are a type of neural networks that are specialized in processing sequential data, such as text [5, Ch. 10].

The translation model has inspired many other works usually called either encoder-decoder [25, 4] or sequence-to-sequence [28, 29] models. The basic idea is to encode an input sequence to a vector and then decode the target sequence from that vector. These models usually make minimal assumptions regarding the sequence structure. The encoder-decoder model has been successfully used in translation tasks and conversation models.

Other versions of this model exist, such as the hierarchical recurrent encoder-decoder neural network by Serban et al. [24]. This model uses two encoder RNNs and one decoder to model a dialog system. One RNN encodes each new utterance and another encodes all utterances seen so far in the conversation to summarize the conversation history. The decoder is then used to predict a response based on the output of the encoders.

Another end-to-end model is the dual-encoder used, among others, by Lowe et al. [18]. The dual-encoder encodes both the context, i.e., the conversation so far, and a possible response. Given the final hidden states from both RNNs, the probability that they are a valid pair is calculated, in other words, how likely the response is given the context.

3.3 End-to-End Trainable Task-Oriented Systems

End-to-end dialog systems have shown promising performance in conversational chatbots. Despite this, it is unclear whether these results would be the same for goal-oriented dialog systems [3] as these kinds of studies do not seem to exist to the same extent in a goal-oriented setting. The reason for this could be the lack of publicly available task-oriented dialog data and common evaluation metrics, which makes it difficult to reproduce experiments and compare the results. One effort to mitigate this problem has been made by Bordes et al. [3] where an open task-oriented dataset was created. Several baseline models were tested on the data. One of the models that generally outperformed the others in this work was a variation of a memory network. Memory networks (MemNNs) were first introduced by Weston et al. [33]. They are a class of learning models originally used for question answering. It has a long-term memory component that can store information. Other variants that require less supervised training have been based on this work, such as the dynamic memory network (DMN) [12] and the end-to-end memory network (MemN2N) [27]. Bordes et al. [3] use the MemN2N architecture from Sukhbaatar et al. [27] with some modifications to leverage exact matches and types.

Another model used in task-oriented dialog is presented by Wen et al. [32]. This model consists of several neural network based components. The components are called intent network, belief tracker, generation network, policy network, and database operator. The intent network handles the input and encodes it into a vector representation. The belief tracker keeps track of an estimated dialog state for specific slots, such as what type of food the user is interested in. The generation network can generate a sentence word by word given a vector describing the state of the system. The policy network binds together the separate modules by combining the output from the intent network, the database, and the belief state. Each module can be trained directly from conversation data except the database operator since the database attributes are explicitly stated. This breaks the differentiability of the system and is the reason why the components have to be trained separately. While the system has an explicit representation of the database attributes, the intent network has a distributed representation of the user intent which allows ambiguous input.

Chapter 4

Theory

The theoretical background that the chatbot implemented in this thesis relies on is presented in this chapter. First, traditional feedforward neural networks are briefly explained as a basis for the next section describing recurrent neural networks (RNNs). Next, two models using RNNs are introduced. The first is sequence-to-sequence learning, a model using RNNs to learn mappings between sequences. This is the model used in this thesis to implement an end-to-end trainable chatbot. The second model is a memory network, which is the most relevant baseline for our results. The chapter is concluded with a discussion about different ways to represent textual input to make training easier for the system.

4.1 Neural Networks

Neural networks are used to approximate some function f [5, Ch. 6]. One common application of a neural network is classification, where some input x is mapped to a class y. This can be described as a mapping $y = f(x; \theta)$ where the network learns the values of the parameters θ that result in the best function approximation [5, Ch. 6].

A traditional feedforward neural network consists of an input and an output layer and possible hidden layers in between [5, Ch. 6]. An example is illustrated in Figure 4.1. The number of nodes in the input layer is determined by the dimensionality of the data. The number of nodes in the output layer is determined by the number of classes or targets. The number of nodes in the hidden layer determines how complex functions the network will be able to learn. Too few nodes may lead to a too simplistic model that cannot learn advanced patterns. An example could be using a linear model for non-linear data. Too many nodes may lead to overfitting the data as well as an increased computational cost in learning the network parameters. An overfitted model performs great on the training data but does not generalize well to new data.

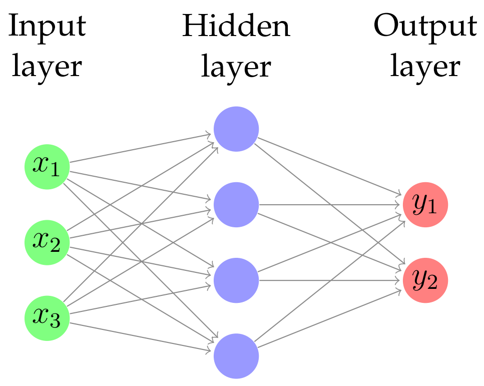

Figure 4.1: A neural network with one hidden layer.

In feedforward networks information propagates forward from the input x to the hidden units at each layer and finally produces an output y [5, Ch. 6]. The information is transformed at each step using matrix multiplications and activation functions [5, Ch. 6]. Each layer has an activation function that transforms the inputs to its outputs. Common activation functions are tanh, sigmoid, or ReLU [5, Ch. 6]. In the output layer, however, the softmax function is commonly used to convert the output to a probability distribution over n possible values [5, Ch. 6]. In the classification case, this corresponds to determining the probability of the input belonging to the different classes.

The goal of a neural network is to find the best function approximation by finding network parameters that minimize the error on our training data [5, Ch. 6]. These parameters are learned through training on labeled data. For each prediction y, the error is measured as compared to the true labels. How the error is measured is defined by a loss or cost function [5, Ch. 6]. The minimum of the loss function can be found using gradient descent [5, Ch. 6]. To calculate the minimum, the gradients of the loss function are needed and can be calculated with backpropagation. Backpropagation is an algorithm to calculate gradients from the output using the chain rule [5, Ch. 6].

4.2 Recurrent Neural Networks

Recurrent neural networks (RNNs) are neural networks that are specialized in processing sequential data, such as text [5, Ch. 10]. Traditional neural networks make the assumption that all inputs and outputs are independent of each other. RNNs instead make the assumption that the inputs depend on each other. This makes the network better at processing sequences of values. In an RNN the neural network is extended to include feedback connections [5, Ch. 10]. At each time step the hidden state of the previous time step is fed back into the network as part of its input. In this way information is preserved over time. The recurrent network can be viewed as a network with a self-loop, as depicted in Figure 4.2. In theory, these networks can handle arbitrarily long sequences. In practice, however, it is common to create an unfolded version of the network, as depicted in Figure 4.3, with a fixed number of steps.

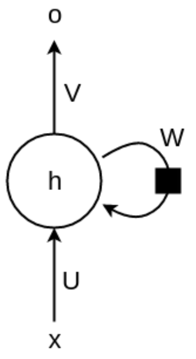

Figure 4.2: Recurrent neural network.

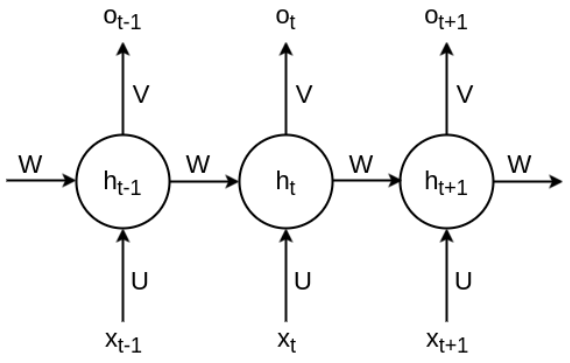

Figure 4.3: Unfolded recurrent neural network.

In Figure 4.2 and Figure 4.3, x is the input at each time step, and o is the output calculated based on the hidden state so that $o_t = f(Vh_t)$ [5, Ch. 10]. The hidden state h is calculated based on the previous hidden state and the current input, i.e., $h_t = f(Ux_t + Wh_{t-1})$ [5, Ch. 10]. U, V, and W are the network parameters to be learned during training. Output and input can be given at each time step but might not always be necessary. Depending on what the network should do there could be one or many inputs and one or many outputs. An example where there are many inputs but only one output is in sentiment analysis, where a sentence is classified as either having a positive or a negative sentiment. The sentence is fed to the network word by word and after the whole sequence of words has been processed the output is the sentiment classification. A case where there are multiple outputs but only one input is when RNNs are used to generate image descriptions. Here the input is an image and the output is a sequence of words describing the image.

In RNNs the network parameters are shared by all time steps [5, Ch. 10]. Because the parameters are shared the gradients depend on previous time steps. To train an RNN an algorithm called backpropagation through time (BPTT) is used [5, Ch. 10]. BPTT works just like the regular backpropagation algorithm on an unrolled RNN.

When using the backpropagation algorithm there are difficulties in learning long-term dependencies [2]. Since backpropagation computes gradients using the chain rule, many small numbers are multiplied together causing the gradient of layers far from the final layer to become extremely small or vanish. There are ways to avoid this problem such as using gated RNNs, e.g., long-short term memories (LSTMs) [9] or the more recent gated recurrent units (GRUs) [4].

LSTMs were first introduced by Hochreiter and Schmidhuber [9] and are a type of RNN that are designed to better handle long-term dependencies. The difference from a regular RNN is how the hidden state is computed. LSTMs have an internal cell state and have the ability to add or remove information from the state. What to add or remove is regulated by structures called gates. Generally, there are three gates called input, forget and output gates. The input and forget gates control the cell state by deciding how much of the new input to add to the state and what to throw away from the cell state. The output gate decides what to output based on the cell state. These gates allow LSTMs to handle long-term dependencies. The LSTM learns how its memory should behave by learning the parameters for the gates. There exist many different versions of the LSTM network. Some of them are evaluated in the work by Greff et al. [6].

4.3 Sequence-to-Sequence Learning

One model that uses recurrent neural networks is the sequence-to-sequence model, sometimes called encoder-decoder model [4]. The model is used to learn a mapping from sequences to other sequences.

This basic idea can be used for many advanced tasks. Two examples are machine translation [28] and conversational models [29]. In machine translation the goal is to map a sequence of words in one language to the corresponding sequence of words in another language. In conversational systems the aim is instead to translate from an input sentence to a system response.

Sequence-to-sequence models exist in many variations but the basic structure consists of two RNNs [4]. One encodes the input into a vector representation capturing the meaning and important information of the input sequence. The other RNN then takes this vector as input and uses it to produce the output sequence. This process is depicted with the unrolled RNNs in Figure 4.4.

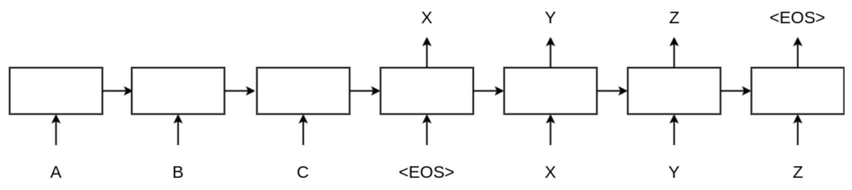

Figure 4.4: The basic architecture of a sequence-to-sequence model.

In the figure, each box is an RNN cell, such as LSTM or GRU. The encoder reads the input, token by token until it gets a special token marking the end of the input. In the figure, this token is $<EOS>$. Based on the context generated by the encoder, the decoder generates the output sentence, token by token until it generates a stop token. At each time step, the previously generated token is fed back into the network as the input.

At each time step the decoder produces a list of probabilities over the vocabulary [28]. These probabilities can be used to either generate a sequence, token by token, or calculate the probability of an existing sequence. To generate a sequence in a greedy manner you select the token with the highest probability at each time step and feed it back into the network to generate the next token [28]. Instead of just considering the most probable token at each step, the top x tokens can be examined. In this way we can use beam-search to find the most probable sequence of tokens. To calculate the probability of an existing sequence given the input sequence we estimate the conditional probability $p(y_1, y_2, ..., y_m|x_1, x_2, ..., x_n)$ [28], where x_i is the input tokens and y_i is the output tokens. In the example in Figure 4.4 this would be $p(X, Y, Z|A, B, C)$. This probability can be rewritten as follows: $p(y_1, ..., y_m|x_1, ..., x_n) = \prod_{i=1}^{m} p(y_i|x_1, ..., x_n, y_1, ..., y_{i-1})$ [28]. Thus we calculate the product of each token in the sequence conditioned on all previous tokens. To select an output sequence from a set of possible sequences one would calculate this probability for all the sequences and select the one with the highest probability.

These two different ways of predicting a response mean that a sequence-to-sequence model can be used to create both a generative and a retrieval based chatbot, either generating or selecting a response. The only difference to the model would be the inference function.

4.4 Memory Networks

Neural networks are good at learning implicit knowledge and patterns but have difficulties remembering facts [5, Ch. 10]. A solution to this is to use memory components to store and retrieve specific facts. Memory has been argued to be important for the ability to reason [5, Ch. 10]. Weston et al. [33] introduced memory networks that use a long-term memory component to store information. This memory can be read and written to and is used in combination with inference components to make predictions. The system learns how to use these components together. Memory networks use hard attention over the memory and require explicit supervision during training [33]. End-to-end memory networks, introduced by Sukhbaatar et al. [27], are an end-to-end trainable version of memory networks. It consists of "a neural network with a recurrent attention model over a possibly large external memory" [27]. This model reads from memory with soft attention and only needs supervision on the final output. The difference between hard and soft attention is that the first use a stochastic mechanism and attend only one region [35]. Soft attention uses a deterministic mechanism and attend to all regions at once, producing a score for each region [35].

4.5 Text Representation

The training data for a chatbot consists of text logs of previous conversations. Many machine learning algorithms require the input to be a fixed length feature vector [13].

There are different ways to represent a word. One simple way is using one-hot vectors. Each word is represented by a vector of multiple 0s and a single 1. The one can, for example, be on the same place in the vector as the word's place in the vocabulary. This approach ensures that every word in the vocabulary is represented by a unique vector. With a large vocabulary these vectors become huge and the data very sparse. This encoding of words does not provide any useful information about the relationships between words.

To limit the dimensionality of the vector space, we can use a technique called embedding. A word embedding is a function that maps words to high-dimensional vectors [1]. This function is often a lookup-table and a matrix with a row for each word in the vocabulary. The matrix is initialized with random vectors for each word and then learns a better representation. There exist many different embedding methods such as the word2vec model by Mikolov et al. [21]. Using a distributed representation of words not only limits the dimensionality but can also encode relationships between words. The words "cat" and "dog" might, for example, get similar vectors since they have similar semantic and grammatical roles [1]. This enables the learning system to leverage what it has learned about something when it processes a similar word.

Chapter 5

Method

This chapter begins with a description of the problem and requirements of our chatbot implementation. After that follows a motivation for and description of the dataset used for training and evaluation. Section 5.3 explains how the system is evaluated. The following sections describe the model, implementation details and how the training was done. The chapter is concluded with a description of how the trained model is used to create an actual chat application.

5.1 Problem Definition

In this thesis, a chatbot was implemented with the goal of assisting users in making a restaurant reservation. A common use case for the application could be when a user wants to find a restaurant with a certain type of food within a specified price range. An example of a possible conversation is shown in Table 5.1. The user does not have to know all the information about what restaurant they want to book beforehand, rather it is the task of the bot to help the user find a good option based on specified preferences.

To achieve its goal the chatbot has to be able to process and interpret natural language input. Based on this input the bot has to perform actions such as asking for more information to clarify a user request, query a knowledge base to retrieve information, interpret the results of queries, and display the results in a suitable way to the user. If the user changes their mind, the bot should be able to recognize this and strive to achieve the new goal instead.

Table 5.1: An example of a conversation between a user and the system. The conversation has the same structure as the conversations in the dataset by Bordes et al. [3]. The system responses are in italic. The rows beginning with a star are results from an API call. The dialog is divided into five tasks that highlight different difficulties and essential capabilities of a chatbot.

Task 1 Issuing API calls	Hi! *Hello, what can I help you with today?* Can you make a restaurant reservation in Rome for eight with Italian cuisine. *I'm on it!* *Which price range are you looking for?* In a cheap price range please. *Ok, let me look into some options for you.* `api_call(Rome, Italian, Eight, Cheap)`
Task 2 Updating API calls	Actually I would prefer for Six. *Sure. Is there anything else to update?* No. *Ok, let me look into some options for you.* `api_call(Rome, Italian, Six, Cheap)`
Task 3 Displaying options	** Restaurant1 R_phone phone1* ** Restaurant1 R_cuisine italian* ** Restaurant1 R_address address1* ** Restaurant1 R_location Rome* ** Restaurant1 R_number six* ** Restaurant1 R_price cheap* ** Restaurant1 R_rating 6* ** Restaurant2 R_phone phone2* ** Restaurant2 R_cuisine italian* ** Restaurant2 R_address address2* ** Restaurant2 R_location Rome* ** Restaurant2 R_number six* ** Restaurant2 R_price cheap* ** Restaurant2 R_rating 4* *What do you think of this option: Restaurant1* No, I don't like that. *Sure, let me find another option for you.* *What do you think of this option: Restaurant2* Let's do it! *Great, let me do the reservation.*
Task 4 Provide information	Can you provide me the address? *Here it is: address2* You rock! *Is there anything else I can help you with?* No thanks. *You're welcome.*
Task 5 Full dialog	

5.2 Data

Data collected from real conversations usually contains various noise such as spelling errors, wrong information, interpretation errors and so on which makes the training more difficult. Larger or more open domains are also more difficult to learn from as the variation of the dialogs are greater and the number of possible answers increases. To limit the difficulties presented by the data itself the chatbot was trained and tested in a very limited domain using mostly synthetically generated data. Using synthetic data allows for more control over the format of the dialog which leads to a more straightforward evaluation of the strengths and limitations of the system. To further test the system in a more realistic setting, data adapted from real human-bot conversations is used.

The conversation data used for training and evaluating the system was created by Bordes et al. [3]. The dataset is an open resource to test end-to-end goal-oriented dialog systems in a restaurant booking domain. The data consists of six different subsets that all aim to test different abilities that are deemed essential for a chatbot in a task-oriented domain. These subsets are referred to as tasks 1–6. Tasks 1–5 consist of synthetically generated dialogs, while task 6 consists of real human-bot conversations. The data used in task 6 is an adaptation of the restaurant reservation dataset from the second dialog state tracking challenge (DSTC2) [8]. The dataset was originally designed to train and test dialog state tracker components. A state tracker keeps track of an estimated dialog state at each step of the conversation in order to choose actions, such as what restaurant to suggest.

The different tasks are described below. An example containing parts of tasks 1–5 is provided in Table 5.1. Examples of each individual task can be found in Appendix A.

- **Task 1: issuing API calls**, tests the system's ability to draft an API call based on interactions with the user. The API call has four required fields: cuisine, location, the number of people, and price range. The user can implicitly state between zero and four of these fields and the system has to learn to ask for the remaining fields.

- **Task 2: updating API calls**, tests the system's ability to handle situations where the user changes their mind about something which requires the system to update an issued API call.

- **Task 3: displaying options**, tests the system's ability to propose options to the user given results from an API call. The user can accept or decline an option, in which case the system must propose a new option. The system should always suggest the restaurant with the highest rating that has not yet been rejected.

- **Task 4: provide extra information**, tests the system's ability to use information from API call results. The user makes a reservation at a restaurant and then asks for additional information about the place. The user can ask for the address or phone number of the place.

- **Task 5: conduct full dialogs**, combines tasks 1–4 to test the system's ability to carry out a complete conversation that contains all difficulties presented by each subtask.

- **Task 6: DSTC2**, also tests the system's ability to conduct full conversations but with data from DSTC2 [8]. The data consists of human-bot conversations. As such, the language and structure of the dialogs are more diverse than in tasks 1–5. The data also contain spelling errors and system faults, such as misunderstanding user input and giving the same restaurant suggestion multiple times.

All tasks are further split into three separate datasets for training, development, and testing. We used the same split as the creators of the dataset [3]. For tasks 1–5, these sets all contain 1,000 dialogs each. Task 6 has more conversations but uses a different split, using more conversations for training and fewer for development. More statistics on the datasets are presented in Table 5.2 and Table 5.3. Tasks 1–5 have two separate test sets, one that uses the same vocabulary as the training set and one that uses words that have not been seen during training to test how well the system can generalize to new tokens. This test is called an out of vocabulary (OOV) test [3]. Both test sets consist of 1,000 dialogs each. Task 6 does not have this kind of test set.

The dataset also has a knowledge base related to tasks 1–5. This knowledge base is a text file containing information about all restaurants that appear in the dataset. The knowledge base contains 1200

Table 5.2: Average number of utterances in each task [3].

	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
Number of utterances	12	17	43	15	55	54
User utterances	5	7	7	4	13	6
Bot utterances	7	10	10	4	18	8
API call outputs	0	0	23	7	24	40

Table 5.3: Number of dialogs in the datasets [3].

	Tasks 1–5	Task 6
Vocabulary size	3,747	1,229
Training dialogs	1,000	1,618
Validation dialogs	1,000	500
Test dialogs	1,000	1,117
OOV test dialogs	1,000	-

unique restaurants. Each restaurant has seven properties: cuisine, location, price range, rating, phone number, address, and the number of available seats. There are 10 different cuisines, 10 locations, 3 price ranges (cheap, moderate, or expensive), and 8 ratings (1–8) with 8 being the best. For simplicity, a restaurant only has room for a party of a single size (2, 4, 6, or 8). An example of an entry in the knowledge base is presented in Table 5.4.

Table 5.4: An example of an entry in the knowledge base. Each restaurant has seven properties. These are cuisine, location, price, rating, phone number, address, and number of available seats.

restaurant_name R_cuisine	korean
restaurant_name R_location	seoul
restaurant_name R_price	cheap
restaurant_name R_rating	1
restaurant_name R_phone	phone
restaurant_name R_address	address
restaurant_name R_number	four

5.3 Evaluation

The goal of the chatbot is to assist the user in booking a table at a restaurant. Its performance was evaluated by measuring the system's ability to predict the next response, given the conversation so far. The correct response in this evaluation is considered to be the response from the actual dialog in the dataset. For example, consider the conversation consisting of the utterances $\{u_1, b_1, u_2, b_2, ..., u_n, b_n\}$, where u_i is a user utterance and b_i is a bot utterance. This conversation will be split into $c_1 = \{u_1\}$, $r_1 = \{b_1\}$, $c_2 = \{u_1, b_1, u_2\}$, $r_2 = \{b_2\}$ and so on, where c_i is the context and r_i is the correct response given that context.

To only consider the response from the actual dialog to be the correct response results in a rather strict evaluation. This might be undesirable if multiple answers can be equally correct. For example, it would be strange to say that "thank you" is incorrect if the correct answer is "thanks". As it turns out, however, we do not have this ambiguity in our dataset. The number of different bot utterances is limited and the bot does not express the same thing in different ways, and thus a response can be considered as correct or incorrect without ambiguity.

The response accuracy is measured both for each response and for the entire conversation. The response accuracy measures the percentage of responses where the bot was able to predict the correct response. The conversation accuracy is calculated as the number of conversations where every response was predicted correctly. Every response is required to be correct because even just one mistake can keep the user from completing their task. These evaluation metrics are the same as in the study by Bordes et al. [3] and the results are compared against the baselines provided in that paper.

5.4 Model

The goal of the system is to predict the correct response given the context. During training the goal is, therefore, to train the model to give the observed response as high probability as possible by maximizing the probability of the observed response given the context, i.e., $p(response|context)$.

Both the context and response can be seen as a sequence of words, which changes the goal to predicting the most likely sequence of tokens given a previously seen sequence of tokens, i.e., maximizing $p(r_1, ..., r_n|c_1, ..., c_m)$. Given this formulation of the system goal, a seemingly fitting model to use is sequence-to-sequence learning [4] (described in Section 4.3). This is a well-known model that exists in many variations and has been used successfully for several natural language tasks.

The responses are generated using the greedy approach described in Section 4.3, creating the response word by word by selecting the word with the highest probability at each step. This differs from how the responses are predicted in the work by Bordes et al. [3]. In their work a response is selected from a set of candidates. This set of candidate responses consists of all bot utterances from the entire dataset (including the test set). The probability of each candidate given the context is calculated and the highest scoring response is selected as the answer.

A retrieval based inference function can be implemented for a sequence-to-sequence model (as mentioned in Section 4.3). The reason for choosing a generative approach instead is mainly because of the simplicity and the computational difference. A retrieval based function has to calculate a score for every possible candidate response to find the one with the highest probability. A generative function creates a response by selecting the highest scoring word at each time step. Thus, a generative approach can require much less computational time, especially if the candidate set is large. There exist ways to speed up a retrieval based approach, such as using clustering techniques to first find a subset of candidate responses to select an answer from [10], but the simplicity of a generative approach made it the preferred option. Even though the prediction is done differently the accuracy results should still be comparable.

5.5 Implementation

The system was implemented using the open-source machine learning library Tensorflow [20]. The sequence-to-sequence model is based on the *tf.contrib.seq2seq* module in Tensorflow 1.0.

Both the encoder and decoder consist of a dynamic RNN, meaning that they put no restriction on the batch size or sequence length.

The regular RNN implementation in Tensorflow internally creates an unrolled version of the RNN with a fixed length, which limits the maximum allowed length of the input to the size first specified. The dynamic RNN implementation instead uses dynamic unrolling of inputs to dynamically construct the computation graph and allow inputs of different sizes. The encoder RNN and the decoder RNN both use LSTM cells. The LSTM implementation is defined in the *tf.contrib.rnn* module and is based on the paper by Hochreiter and Schmidhuber [9]. In their implementation, each cell can have a number of units. This refers to the size of the hidden state of the cell and decides the network's learning capacity. Using more units makes it possible to memorize more of the training data, but it will take longer to train and have a risk of overfitting. Between 50 and 300 units were used in the different tests. Both encoder and decoder had the same number of units. More units did generally not give significantly better or worse results but did slow down the training.

The conversation data was parsed into context-response pairs as described in Section 5.3. These strings are tokenized and each word assigned an index. These lists of numbers are the input and output of the model. The model then embeds the numbers into vector representations. The embeddings are learned during training.

5.6 Training

The model was trained using weighted cross-entropy loss for sequences of logits using the Tensorflow function *sequence_loss* defined in the module *tf.contrib.seq2seq*. The loss is minimized using the Adam optimizer as described by Kingma and Ba [11]. Adam is "an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments" [11]. The Tensorflow implementation defined in the *tf.train* module was used with a learning rate of 0.001.

The model was trained for several epochs, i.e., several runs over all training examples. After each epoch, the loss and accuracy of the model were calculated on the validation dataset. The training was terminated when the accuracy no longer improved. The validation dataset was also used to identify parameters of the model such as embedding size and number of units in the encoder and decoder. The model that achieved the best accuracy on the validation set was then used to calculate the accuracy on the test sets.

5.7 Chat Program

We can use a pre-trained model to create a chat application capable of interpreting user input and generate responses. The pre-trained model is loaded into the program and the user input is fed into it. Given this input, the model makes a prediction by generating a sentence, word by word. The user input and system response are then added to a context set to keep track of what has been said. For practical purposes, the size of the context is limited and the chatbot will start to "forget" things that were said too far back. This limit can be adjusted to let the system have a longer or shorter memory. In our implementation we used a limit equal to the longest conversation in the training data. In the next step, the model makes a prediction given this context combined with the next user input, and so on.

The system can formulate API calls that are used to query a knowledge base. In our implementation, we use the knowledge base provided with the dataset. This knowledge base contains information about all restaurants that appear in the dataset. If the system generates a response that starts with the token *api_call* we interpret it as an API call of the format *api_call <cuisine> <location> <number> <price>* and search through the knowledge base for matches. The matching strings are then added to the context. This means that if we were to use a model trained on a different dataset we would need a corresponding knowledge base as well. Without a knowledge base, all API calls will return nothing.

Chapter 6

Results

The accuracy results of our model are presented in the last column (seq-to-seq) of Table 6.1. On task 1 (issuing API calls) and task 2 (updating API calls), our sequence-to-sequence model achieves 100% accuracy which means that every response was predicted correctly. When testing using words that were not seen during training, the accuracy per response sunk to around 80% and the accuracy per conversation to 0%. For both tests, the errors on the out of vocabulary test set were almost exclusively API calls with the wrong entity, i.e., "api_call italian madrid two expensive" instead of "api_call thai tokyo two expensive". Examples of prediction errors can be found in Appendix B. For task 3 (displaying options) and task 4 (provide extra information), the accuracy is almost the same for both the regular test set and the out of vocabulary test set. On both tasks, we get 0% per conversation accuracy. Most errors are again caused by the use of the wrong entity. The errors, on task 3, have the form "what do you think of this option wrong_resturant". The results from an API call, that are present in the context, are often ignored and some restaurant from the training data is used instead. On task 4, all errors occur when the user asks for information about a restaurant, i.e., an address or phone number. The system either gives the wrong information or none at all.

Table 6.1: Accuracy results for several different models from Bordes et al. [3] combined with results from our sequence-to-sequence model in the last column. The accuracy of each model is given as percentages per response and per conversation (shown in parenthesis). Best performing learning models, considering per response accuracy, are marked in bold. If the difference is less than 0.1% both results are marked. The test sets that use words that are not in the training data are marked by OOV (out of vocabulary).

	Rule-based Systems	TF-IDF Match	Nearest Neighbor	Supervised Embeddings	Memory Networks		Seq-to-Seq
						+ Match Type	
Task 1	100 (100)	5.6 (0)	55.1 (0)	**100** (100)	**99.9** (99.6)	**100** (100)	**100** (100)
Task 2	100 (100)	3.4 (0)	68.3 (0)	68.4 (0)	**100** (100)	98.3 (83.9)	**100** (100)
Task 3	100 (100)	8.0 (0)	58.8 (0)	64.9 (0)	**74.9** (2.0)	**74.9** (0)	**74.9** (0)
Task 4	100 (100)	9.5 (0)	28.6 (0)	57.2 (0)	59.5 (3.0)	**100** (100)	57.2 (0)
Task 5	100 (100)	4.6 (0)	57.1 (0)	75.4 (0)	96.1 (49.4)	93.4 (19.7)	**99.0** (85.0)
Task 6	33.3 (0)	1.6 (0)	21.9 (0)	22.6 (0)	41.1 (0)	41.0 (0)	**47.1** (1.8)
Task 1 (OOV)	100 (100)	5.8 (0)	44.1 (0)	60.0 (0)	72.3 (0)	**96.5** (82.7)	81.1 (0)
Task 2 (OOV)	100 (100)	3.5 (0)	68.3 (0)	68.3 (0)	78.9 (0)	**94.5** (48.4)	78.9 (0)
Task 3 (OOV)	100 (100)	8.3 (0)	58.8 (0)	65.0 (0)	74.4 (0)	**75.2** (0)	**75.3** (0)
Task 4 (OOV)	100 (100)	9.8 (0)	28.6 (0)	57.0 (0)	57.6 (0)	**100** (100)	57.0 (0)
Task 5 (OOV)	100 (100)	4.6 (0)	48.4 (0)	58.2 (0)	65.5 (0)	**77.7** (0)	66.9 (0)

On task 5 (conduct full dialogs), our system achieved 99% per response accuracy and 85% per conversation. Similar to previous tasks, task 5 makes errors when using entities. For the regular test set, all errors were caused by providing the wrong address or phone number to a restaurant. On the out of vocabulary test we still have issues with using the correct entity and making errors when issuing API calls, providing information, and suggesting restaurants. Also, we make some other errors where the first word of the sentence is wrong, e.g., "where price range are you looking for" and "any many people would be in your party".

The errors on task 6 are more diverse. Commonly the prediction is close to the expected answer but with a few wrong words. In other cases the prediction is completely nonsensical, such as "would are restaurants in the expensive of town do tasty kind of food would area range".

The accuracy of our sequence-to-sequence model is compared against several baseline systems, as shown in Table 6.1. The baseline results are taken from Bordes et al. [3]. The baseline models consist of a rule-based system, two classical information retrieval models, and two network based models. For more details on the implementation of the baseline models, please refer to the paper by Bordes et al. [3].

The sequence-to-sequence model outperforms the traditional information retrieval models on all tasks. It also gets equal or better results compared to the supervised embedding model. The accuracy of our model is similar to that of the memory network without match type features. With match type features, memory networks achieve the best result on most tasks, the only exceptions being tasks 5 and 6 where our model performs slightly better. Match type features seem to improve the result particularly on the out of vocabulary tests. The rule-based system achieves 100% accuracy on tasks 1–5 on both test sets, but has worse results on task 6.

Chapter 7

Discussion

Our sequence-to-sequence model can achieve high per response accuracy that is comparable to the best performing baseline model. The biggest problem the system has, and which causes most of the errors, is exact entity matches. The sequence-to-sequence model can learn to model language and answer in a predictable way but does not perform well when a specific entity is required, such as a restaurant name or an address. Most of the time it recognizes what is being asked for, for example, a phone number, a restaurant name, or an address, but it responds with the wrong one. This indicates that the system can learn the pattern but has a hard time distinguishing between entities of the same type. Our model also has trouble dealing with entities that have not been seen during training and simply does not use them, as can be seen by the fact that the per conversation accuracy is 0% on all out of vocabulary tests. The reason we still get relatively high per response accuracies is that the system gets almost all other responses correct, i.e., those responses containing words actually seen during training. This limits the usefulness of the chatbot to known entities. To, for example, add a new restaurant it would not be enough to add it to the knowledge base. We would also have to provide the chatbot with new training data on how to use it. This shortcoming is not something we can ignore as it is a crucial ability for a task-oriented chatbot. A restaurant reservation system is not usable if it cannot differentiate between different restaurants, addresses, phone numbers, and so on.

The model achieves great results on task 1 and task 2 with 100% accuracy on both. On task 3 and task 4, however, the per conversation accuracy drastically drops to 0%. This is caused by the system's difficulty with using results from API calls to suggest restaurants and provide information about them. For these two tasks it does not seem to make a difference if the system has seen the entities during training or not, as we get 0% conversation accuracy on both the OOV tests and the regular tests. Both task 3 and task 4 struggle with the same shortcoming of the system but get quite different per response accuracies. This is most likely because of the different lengths of the conversations in both tasks. As presented in Table 5.3, task 3 consists of conversations with 17 utterances on average while task 4 has an average of eight (not counting results from API calls). On task 5, where we conduct full dialogs, the system still makes the same errors but much more rarely.

Rule-based systems have the best performance on most tasks. This is probably because tasks 1–5 are synthetically generated, have a simple structure, and are thus predictable and not too difficult to create rules for that can achieve 100% accuracy. On task 6, that consists of human-bot conversations, both memory networks and our sequence-to-sequence model outperform the rule-based system. This shows that neural networks are useful in real-life situations where the input is more varied and the conversations are less structured. A network can learn to deal with situations that we have a hard time predicting in advance, and thus creating rules for. This suggests that machine learning is useful in real situations where we have non-synthetic or unstructured data.

While performing better than the rule-based system on task 6, our model still achieves an accuracy of less than 50%. Being able to answer correctly less than half of the time is not sufficient for use in a real-world application. Even one error in a conversation, such as making the incorrect API call, can prevent the user from completing their task. A per conversation accuracy of 1.8% is clearly unacceptable. The results do however indicate that rule-based and network-based systems might have different strengths and weaknesses. Thus a combination of the two might be a good idea. We could use rules for situations that are easy to predict and write rules for, while the network can learn to deal with unexpected situations or situations where the input is hard to define with a rule-based system. This could be to learn to recognize greetings, to find entities in a request, and so on.

Chapter 8

Conclusions and Future Work

A task-oriented conversation system was implemented using a sequence-to-sequence model. The model was trained end-to-end on restaurant reservation data and compared against several baselines. Our model can, without any previous knowledge of the domain, learn to formulate sentences in order to conduct basic conversations. It can successfully formulate API calls and update those calls if the user wants to change something.

The most noticeable issue with the system is that it has troubles using the correct entities when exact matches are required. The system has difficulties learning to use results of API calls and favors responses seen during training instead of using entities from the context. Words that have not been seen during training are never used which means that the system cannot handle new entities without retraining. The system's failure on these critical tasks makes it unsuitable to use as a standalone chatbot. These tests have, however, shown some strengths and limitations of a sequence-to-sequence model in the context of task-oriented conversations. This knowledge can be used for future testing and development.

The purpose of this thesis was to investigate to what extent a restaurant reservation system can be automated using an end-to-end trainable model. The tests have shown some promising strengths of the implemented system but also significant flaws that cannot be overlooked. In conclusion, our current model cannot be used as a standalone system to successfully conduct full conversations with the goal of making a restaurant reservation. The review has however contributed with a thorough examination of the failures of such a system and shown where future work might make the most difference. The results could also provide a baseline for future work.

8.1 Future Work

The work in this thesis was limited to a rather small domain. Tests on other, perhaps broader, domains would be needed to see how the results scale. It would also be of interest to try additional learning models and assess their performance in comparison to the models discussed in this thesis. There exist many different neural network based conversational models that each have their own strengths and weaknesses.

There are some possible modifications to the sequence-to-sequence model that would be interesting to implement and see if they improve the results. One modification would be to try to encourage exact matches, as the match type feature used by Bordes et al. [3]. Another modification would be to replace the greedy-search in the response generation with a more advanced algorithm, such as beam-search.

The inference function currently generates responses word by word. One possible improvement is to implement a retrieval based response generation. A retrieval based inference function should be able to improve the accuracy and allow more control over the answers. Thereby we could avoid the nonsensical and grammatically wrong sentences we have now. The drawback is the limitations on the possible answers and the extra effort needed to create the answer set. There is also an added computational cost.

The rule based systems outperformed the network based models on the simpler more structured tasks 1–5 while the network based models performed better on the more unstructured task 6. Therefore, it would be interesting to make a hybrid model that combines the strengths of both approaches. In some cases all we really need is to match a certain word while other times the task is more difficult. If we start making rules based on our domain knowledge it would, however, remove the end-to-end trainability and the portability of the system. This might, however, be a reasonable approach for implementing a chatbot system today as the networks are far from accurate enough to work by themselves at the moment.

In chat applications, more than just the accuracy is important. How the user feels when using the system is just as important. The chat application is, however, rather minimal at the moment. As the goal of this thesis was to evaluate the power of using neural networks to create a chatbot, the application was made using almost solely the sequence-to-sequence model and nothing else. Therefore there are many improvements that can lead to enhanced user experience. One simple improvement is to calculate the probability of our response at each step of the conversation and give the user a warning if the probability is below some threshold. This message could be of the form "Sorry, I'm not certain that I understood your request but my best guess is: ...". This limits the damage if the system is wrong and gives the user a hint of what kind of questions the system can answer with certainty and which it cannot. This threshold has to be set appropriately so that the message does not show too often and creates additional annoyance instead of being helpful. Sometimes the chatbot responds in a way that did not further the user towards their goal. When this faulty response is entered into the context it could confuse the chatbot when making future predictions. A way to combat this could be to detect when we made an error and not append that part to the context. An alternative is to train the system to recover from this state by training it on these types of conversations as well.

Above we have mentioned some ideas that build on our findings in this thesis but there are of course much more possibilities for future work in this research area.

Bibliography

[1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. "A Neural Probabilistic Language Model". In: *Journal of Machine Learning Research* (2003), pp. 1137–1155.

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning Long-Term Dependencies with Gradient Descent is Difficult". In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. DOI: 10.1109/72.279181.

[3] Antoine Bordes, Y-Lan Boureau, and Jason Weston. "Learning End-to-End Goal-Oriented Dialog". In: *CoRR* abs/1605.07683v3 (2016). URL: https://arxiv.org/abs/1605.07683v3.

[4] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). URL: http://arxiv.org/abs/1406.1078.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[6] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. "LSTM: A Search Space Odyssey". In: *IEEE Transactions on Neural Networks and Learning Systems* (2017), pp. 1–11. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2016.2582924.

[7] Bruce Hanington and Bella Martin. *Universal Methods of Design: 100 Ways to Research Complex Problems, Develop Innovative Ideas, and Design Effective Solutions*. Rockport Publishers, 2012.

[8] Matthew Henderson, Blaise Thomson, and Jason D. Williams. "The Second Dialog State Tracking Challenge". In: *Proceedings of SIGdial*. 2014.

[9] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[10] Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann, Andrew Tomkins, Balint Miklos, Greg Corrado, László Lukács, Marina Ganea, Peter Young, and Vivek Ramavajjala. "Smart Reply: Automated Response Suggestion for Email". In: *CoRR* abs/1606.04870 (2016). URL: `http://arxiv.org/abs/1606.04870`.

[11] Diederik P. Kingma and Jimmy Lei Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). URL: `http://arxiv.org/abs/1412.6980`.

[12] Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. "Ask Me Anything: Dynamic Memory Networks for Natural Language Processing". In: *CoRR* abs/1506.07285 (2015). URL: `http://arxiv.org/abs/1506.07285`.

[13] Quoc V. Le and Tomas Mikolov. "Distributed Representations of Sentences and Documents". In: *ICML*. Vol. 14. 2014, pp. 1188–1196.

[14] Oliver Lemon, Kallirroi Georgila, James Henderson, and Matthew Stuttle. "An ISU Dialogue System Exhibiting Reinforcement Learning of Dialogue Policies: Generic Slot-filling in the TALK In-car System". In: *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics: Posters & Demonstrations*. EACL '06. Trento, Italy: Association for Computational Linguistics, 2006, pp. 119–122.

[15] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. "A Diversity-Promoting Objective Function for Neural Conversation Models". In: *CoRR* abs/1510.03055 (2015). URL: `http://arxiv.org/abs/1510.03055`.

[16] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. "A Persona-Based Neural Conversation Model". In: *CoRR* abs/1603.06155 (2016). URL: `http://arxiv.org/abs/1603.06155`.

[17] Chia-Wei Liu, Ryan Lowe, Iulian V. Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. "How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation". In: *CoRR* abs/1603.08023 (2016). URL: `http://arxiv.org/abs/1603.08023`.

[18] Ryan Lowe, Nissan Pow, Iulian V. Serban, Laurent Charlin, Chia-Wei Liu, and Joelle Pineau. "Training End-to-End Dialogue Systems with the Ubuntu Dialogue Corpus". In: *Dialogue & Discourse* 8.1 (2017), pp. 31–65.

[19] Ryan Lowe, Iulian V. Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. "On the Evaluation of Dialogue Systems with Next Utterance Classification". In: *CoRR* abs/1605.05414 (2016). URL: `http://arxiv.org/abs/1605.05414`.

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

[21] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeff Dean. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems*. 2013, pp. 3111–3119.

[22] 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*. ACL '02. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pp. 311–318. DOI: 10.3115/1073083.1073135.

[23] 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*. EMNLP '11. Edinburgh, United Kingdom: Association for Computational Linguistics, 2011, pp. 583–593. ISBN: 978-1-937284-11-4.

[24] Iulian V. Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. "Building End-to-end Dialogue Systems Using Generative Hierarchical Neural Network Models". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI'16. Phoenix, Arizona: AAAI Press, 2016, pp. 3776–3783.

[25] Lifeng Shang, Zhengdong Lu, and Hang Li. "Neural Responding Machine for Short-Text Conversation". In: *CoRR* abs/1503.02364 (2015). URL: http://arxiv.org/abs/1503.02364.

[26] 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". In: *CoRR* abs/1506.06714 (2015). URL: http://arxiv.org/abs/1506.06714.

[27] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. "End-To-End Memory Networks". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems*. NIPS'15. Montreal, Canada: MIT Press, 2015, pp. 2440–2448.

[28] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems*. 2014, pp. 3104–3112.

[29] Oriol Vinyals and Quoc V. Le. "A Neural Conversational Model". In: *CoRR* abs/1506.05869 (2015). URL: `http://arxiv.org/abs/1506.05869`.

[30] Zhuoran Wang and Oliver Lemon. "A Simple and Generic Belief Tracking Mechanism for the Dialog State Tracking Challenge: On the believability of observed information". In: *Proceedings of the SIGDIAL 2013 Conference*. Metz, France: Association for Computational Linguistics, Aug. 2013, pp. 423–432.

[31] Joseph Weizenbaum. "ELIZA–a Computer Program for the Study of Natural Language Communication Between Man and Machine". In: *Communications of the ACM* 9.1 (Jan. 1966), pp. 36–45. DOI: `10.1145/365153.365168`.

[32] Tsung-Hsien Wen, Milica Gašić, Nikola Mrksic, Lina Maria Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve Young. "A Network-based End-to-End Trainable Task-oriented Dialogue System". In: *CoRR* abs/1604.04562 (2016). URL: `http://arxiv.org/abs/1604.04562`.

[33] Jason Weston, Sumit Chopra, and Antoine Bordes. "Memory Networks". In: *CoRR* abs/1410.3916 (2014). URL: `http://arxiv.org/abs/1410.3916`.

[34] Jason D. Williams and Steve Young. "Partially observable Markov decision processes for spoken dialog systems". In: *Computer Speech & Language* 21.2 (2007), pp. 393–422. DOI: `10.1016/j.csl.2006.06.008`.

[35] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: *International Conference on Machine Learning*. 2015, pp. 2048–2057.

[36] Steve Young, Milica Gašić, Blaise Thomson, and Jason D. Williams. "POMDP-Based Statistical Spoken Dialog Systems: A Review". In: *Proceedings of the IEEE* 101.5 (May 2013), pp. 1160–1179.

Appendix A

Example Dialogs

This appendix contains examples adapted from the restaurant reservation dataset by Bordes et al. [3]. Tables A.1–A.5 show examples of dialogs from tasks 1–5. These examples show the structure and goal of each task. The dialogs in task 3 and task 4 begin with the result of an API call for the system to use in the task. The results are provided to focus the test on the system's ability to handle the results of an API call while task 1 and task 2 focus on making the correct API call. Task 5 then combines these different tasks to test the system's ability to conduct full conversations. Task 6, like task 5, consists of complete conversations. The data comes from human-bot conversations [8]. As such, the conversations are more diverse. Tables A.6–A.8 show different conversations from the training data. In all examples special <*silence*> tokens are used to mark when the user does not provide further input. The API call results and system responses are in italic.

Table A.1: An example of a dialog from task 1 (issuing API calls) [3].

hi
hello what can i help you with today
can you book a table for two with indian food
i'm on it
<silence>
where should it be
madrid please
which price range are you looking for
i am looking for a cheap restaurant
ok let me look into some options for you
<silence>
api_call indian madrid two cheap

Table A.2: An example of a dialog from task 2 (updating API calls) [3].

hi
hello what can i help you with today
i'd like to book a table in a cheap price range for six people with spanish cuisine in madrid
i'm on it
<silence>
ok let me look into some options for you
<silence>
api_call spanish madrid six cheap
instead could it be for four
sure is there anything else to update
actually i would prefer in rome
sure is there anything else to update
no
ok let me look into some options for you
<silence>
api_call spanish rome four cheap
thanks
you're welcome

Table A.3: An example of a dialog from task 3 (displaying options) [3]. The dialog starts with results from an API call for the system to use in the task.

the_nice_place R_phone phone1
the_nice_place R_cuisine italian
the_nice_place R_address address1
the_nice_place R_location madrid
the_nice_place R_number two
the_nice_place R_price expensive
the_nice_place R_rating 4
the_fancy_place R_phone phone2
the_fancy_place R_cuisine italian
the_fancy_place R_address address2
the_fancy_place R_location madrid
the_fancy_place R_number two
the_fancy_place R_price expensive
the_fancy_place R_rating 7

hello
hello what can i help you with today
i'd like to book a table in madrid with italian cuisine for two in a expensive price range
i'm on it
<silence>
ok let me look into some options for you
<silence>
what do you think of this option: the_fancy_place
no this does not work for me
sure let me find an other option for you
<silence>
what do you think of this option: the_nice_place
i love that
great let me do the reservation

Table A.4: An example of a dialog from task 4 (provide extra information) [3]. The dialog starts with results from an API call for the system to use in the task.

restaurant1 R_phone phone1
restaurant1 R_cuisine french
restaurant1 R_address address1
restaurant1 R_location madrid
restaurant1 R_number two
restaurant1 R_price moderate
restaurant1 R_rating 1
hi
hello what can i help you with today
can you book a table at restaurant1
great let me do the reservation
can you provide the address
here it is address1
may i have the phone number of the restaurant
here it is phone1

Table A.5: An example of a dialog from task 5 (conduct full dialogs) [3].

good morning
hello what can i help you with today
i'd like to book a table for two in london in a moderate price range
i'm on it
<silence>
any preference on a type of cuisine
with spanish cuisine
ok let me look into some options for you
<silence>
api_call spanish london two moderate
instead could it be with british food
sure is there anything else to update
no
ok let me look into some options for you
<silence>
api_call british london two moderate
restaurant1 R_phone phone1
restaurant1 R_cuisine british
restaurant1 R_address address
restaurant1 R_location london
restaurant1 R_number two
restaurant1 R_price moderate
restaurant1 R_rating 5
...
<silence>
what do you think of this option: restaurant1
i love that
great let me do the reservation
what is the phone number of the restaurant
here it is phone1
thank you
is there anything i can help you with
no thank you
you're welcome

Table A.6: A dialog from the training data in task 6 (DSTC2) [3].

<silence>
Hello, welcome to the Cambridge restaurant system. You can ask for restaurants by area, price range or food type. How may I help you?
i want a moderately priced restaurant that serves
What kind of food would you like?
canapes food
What part of town do you have in mind?
it doesnt matter
api_call canapes R_location moderate
api_call no result
<silence>
yippee_noodle_bar is a nice restaurant in the centre of town in the moderate price range
gastropub food
Sorry there is no canapes restaurant in the moderate price range
i want a gastropub restaurant
the cow pizza kitchen and bar is a nice restaurant in the centre of town serving gastropub food
what is the address
Sure, the cow pizza kitchen and bar is on the_cow_pizza_kitchen_and_bar_address
what is the phone number
The phone number of the cow pizza kitchen and bar is the_cow_pizza_kitchen_and_bar_phone
what is the area
the cow pizza kitchen and bar is a nice place in the centre of town
thank you goodbye
you are welcome

Table A.7: A dialog from the training data in task 6 (DSTC2) [3]. In this conversation, the system fails to suggest a new restaurant to the user.

<silence>
Hello, welcome to the Cambridge restaurant system . You can ask for restaurants by area, price range or food type. How may I help you ?
i want a moderately priced restaurant in the west part of town
api_call R_cuisine west moderate
saint_johns_chop_house R_post_code saint_johns_chop_house_post_code
saint_johns_chop_house R_cuisine british
saint_johns_chop_house R_location west
saint_johns_chop_house R_phone saint_johns_chop_house_phone
saint_johns_chop_house R_address saint_johns_chop_house_address
saint_johns_chop_house R_price moderate
saint_johns_chop_house R_rating 3
prezzo R_post_code prezzo_post_code
prezzo R_cuisine italian
prezzo R_location west
prezzo R_phone prezzo_phone
prezzo R_address prezzo_address
prezzo R_price moderate
prezzo R_rating 9
<silence>
prezzo is a nice restaurant in the west of town in the moderate price range
is there anything else
You are looking for a restaurant is that right?
give me a different restaurant
prezzo is a nice restaurant in the west of town in the moderate price range
goodbye
you are welcome

Table A.8: A dialog from the training data in task 6 (DSTC2) [3]. In this example, the user does not use full sentences.

<silence>
Hello, welcome to the Cambridge restaurant system . You can ask for restaurants by area, price range or food type. How may I help you?
restaurant any area kosher food
What kind of food would you like?
kosher
api_call kosher R_location R_price
api_call no result
<silence>
cote is a nice restaurant in the centre of town serving french food
type of food
cote is a great restaurant
phone number
The phone number of cote is cote_phone
thank you good bye
you are welcome

Appendix B

Prediction Errors

This appendix contains examples of different prediction errors that are commonly made by our system. Table B.1 shows that the system has difficulties using entities that were not present in the training data. If the system produces strange sentences it is often the first word that is wrong, as shown in Table B.2. Table B.3 and Table B.4 demonstrate the difficulty at using the results from API calls. Tables B.5–B.7 show dialogs from task 6. In all examples special <*silence*> tokens are used to mark when the user does not provide further input. The API call results and system responses are in italic.

Table B.1: Prediction error made by the system on the out of vocabulary test set of task 1. The words "thai" and "tokyo" have not been seen during training and thus the system favors the words "italian" and "madrid" instead.

context:	hello *hello what can i help you with today* may i have a table *i'm on it* <silence> *any preference on a type of cuisine* with thai cuisine *where should it be* tokyo please *how many people would be in your party* for two people please *which price range are looking for* i am looking for a expensive restaurant *ok let me look into some options for you* <silence>
correct:	api_call thai tokyo two expensive
prediction:	api_call italian madrid two expensive

Table B.2: Prediction error made by the system on the out of vocabulary test set of task 1. Only the first word of the prediction is wrong.

context:	hello *hello what can i help you with today* may i have a table in seoul *i'm on it* <silence> *any preference on a type of cuisine* with thai cuisine
correct:	how many people would be in your party
prediction:	where many people would be in your party

Table B.3: Prediction error made by the system on task 3. The system does not use the provided API call results and suggests the wrong restaurant.

context:	*resto_rome_moderate_british_8stars r_phone phone1* *resto_rome_moderate_british_8stars r_cuisine british* *resto_rome_moderate_british_8stars r_address address1* *resto_rome_moderate_british_8stars r_location rome* *resto_rome_moderate_british_8stars r_number two* *resto_rome_moderate_british_8stars r_price moderate* *resto_rome_moderate_british_8stars r_rating 8* *resto_rome_moderate_british_4stars r_phone phone2* *resto_rome_moderate_british_4stars r_cuisine british* *resto_rome_moderate_british_4stars r_address address2* *resto_rome_moderate_british_4stars r_location rome* *resto_rome_moderate_british_4stars r_number two* *resto_rome_moderate_british_4stars r_price moderate* *resto_rome_moderate_british_4stars r_rating 4* hello *hello what can i help you with today* can you book a table in rome in a moderate price range *i'm on it* <silence> *any preference on a type of cuisine* with british food *how many people would be in your party* for two please *ok let me look into some options for you* <silence>
correct:	what do you think of this option resto_rome_moderate_british_8stars
prediction:	what do you think of this option resto_rome_moderate_french_7stars

Table B.4: Prediction error made by the system on task 4. The system fails to provide the information asked for.

context:	*resto_hanoi_expensive_japanese_1stars r_phone resto_hanoi_expensive_japanese_1stars_phone* *resto_hanoi_expensive_japanese_1stars r_cuisine japanese* *resto_hanoi_expensive_japanese_1stars r_address resto_hanoi_expensive_japanese_1stars_address* *resto_hanoi_expensive_japanese_1stars r_location hanoi* *resto_hanoi_expensive_japanese_1stars r_number six* *resto_hanoi_expensive_japanese_1stars r_price expensive* *resto_hanoi_expensive_japanese_1stars r_rating 1* hello *hello what can i help you with today* can you make a restaurant reservation at resto_hanoi_expensive_japanese_1stars *great let me do the reservation* may i have the phone number of the restaurant
correct:	here it is resto_hanoi_expensive_japanese_1stars_phone
prediction:	here it is

Table B.5: Prediction error made by the system on task 6. The system understands that a phone number is asked for but has difficulties using the correct entities in sentences.

context:	<silence> *hello welcome to the cambridge restaurant system you can ask for restaurants by area price range or food type how may i help you* i would like to find a restaurant that serves halal food *you are looking for a halal restaurant right* halal *i m sorry but there is no restaurant serving halal food* is there a portugeuse food *could you please repeat that* how about portugeuse food *there are restaurants serving portuguese food what area do you want* any area *there are restaurants serving portuguese food in any part of town what price range would you like* any price range *api_call halal r_location r_price* *api_call no result* <silence> *nandos serves portuguese food* whats the phone number
correct:	the phone number of nandos is nandos_phone
prediction:	the phone number of nandos_city_centre is nandos_phone

Table B.6: Prediction error made by the system on task 6. The prediction only differs from the correct response in the first word and could be equally correct.

context:	<silence> *hello welcome to the cambridge restaurant system you can ask for restaurants by area price range or food type how may i help you* <silence> *what kind of food would you like* persian restaurant in the north part of town *i m sorry but there is no restaurant serving persian food* chinese food *could you please repeat that* chinese food
correct:	did you say you are looking for a restaurant in the north of town
prediction:	would you say you are looking for a restaurant in the north of town

Table B.7: Prediction error made by the system on task 6. The prediction is a nonsensical sentence.

context:	<silence> *hello welcome to the cambridge restaurant system you can ask for restaurants by area price range or food type how may i help you* <silence> *what kind of food would you like* i dont care in the north part of town
correct:	there are restaurants in the north of town serving any kind of food what price range would you like
prediction:	would are restaurants in the expensive of town do tasty kind of food would area range