Master Thesis

A retrieval-based Dialogue System utilizing utterance and context embeddings

Alexander Bartl

Master Thesis [DKE 17-07]

Thesis submitted in partial fulfillment
of the requirements for the degree of Master of Science
of Artificial Intelligence at the Department of
Data Science and Knowledge Engineering
of the Maastricht University

Thesis Committee:

Dr. G. Spanakis, Dr. K. Driessens

Maastricht University
Faculty of Humanities and Sciences
Department of Data Science and Knowledge Engineering
Master Artificial Intelligence

April 10, 2017

Abstract

Finding semantically rich and computer-understandable representations for textual dialogues, utterances and words is crucial for dialogue systems or conversational agents, as their performance mostly depends on understanding the context of conversations. Recent research aims at finding distributed vector representations (embeddings) for words, such that semantically similar words are related close within the vector-space. Encoding the "meaning" of text into vectors is a current trend, and the text can range from words, phrases and documents to actual human-to-human conversations. In related research, responses have been generated utilizing a decoder architecture, given the vector representation of the current conversation. In this thesis, the utilization of embeddings for answer retrieval is explored by using Locality-Sensitive Hashing Forest (LSH Forest), an Approximate Nearest Neighbor (ANN) model, to find similar conversations in a corpus and rank possible candidates. Experimental results on a Dutch customer service chat dataset show that, in combination with a candidate selection method, retrieval-based approaches outperform generative ones and reveal promising future research directions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Text-only based Dialogue systems, also called Conversational Agents, Chatbots or Chatterbots, have become more popular in the research community and large companies. The reason for the rise in popularity lies in the fact that their ability to interact intelligently with humans has improved significantly due to advancements in hardware technologies and Artificial Intelligence, for example consider Xiaoice [Markoff and Mozur, 2015], a chatbot designed by Microsoft for the Chinese community. With a design, focused on an appealing personality, the chatbot reached an impressive amount of 20 million registered users with on average 60 interactions per month. Amazon's Alexa, IBM's Watson, Apple's Siri and Google Now are other examples of chatbots developed by major companies, designed to assist humans in their daily tasks or to provide access to their compnay's services.

In contrast to earlier approaches, the newest chatbots may rely on the latest machine learning techniques or heavy scripting to implement useful behaviour. ELIZA [Weizenbaum, 1966], one of the first chatbots, did not aim to find out the overall meaning of a conversation or question, but rather tried to mimic comprehension by spotting certain keywords or key phrases and give predefined answers. Such a system would appear intelligent at first but would lose its impression as soon as its response patterns become apparent. For more comprehensive interactions, it would be necessary to find a computable representation of the meaning of a conversation and give a suitable answer based on this understanding, with the domain the agent is placed in more or less defining what is 'suitable'. For instance, given a question like *"why are my invoices always so high?"*, a chatbot performing as a customer-service agent might try to help the customer by asking for the last contractual changes. In contrast, a chatbot designed for chitchat might answer on a more personal level (*"you always spend too much..."*).

A difficulty in implementing algorithms that were built on the understanding of text is that the meaning of words or word sequences has to be encoded into a representation that is algorithmically usable. Similar to a non-reader that learns to read, plain words have to be mapped to concepts.

One of the latest effective approaches [Mikolov et al., 2013b, Serban et al., 2016b, Serban et al., 2016c] is to represent words, phrases, or even complete dialogues as fixed-length vectors of floating point numbers, where vectors representing text with similar meaning are relatively close to each other. Such vectors, also called embeddings, distributed representations, or feature vectors, have lengths that are adjusted to the target they represent: long vectors to represent many words, short vectors for fewer or single words.

Skip-gram models [Mikolov et al., 2013a], Noise Contrastive Estimation (NCE) [Mikolov et al., 2013b], and other techniques have lately been used to generate word embeddings. The vectors are usually obtained through a training process that tries to optimize an objective function that measures how well consecutive words can be predicted, given the current word vectors. Throughout the training, the embeddings change in a way that favors dense high-quality features, letting them become better at representing the words they are associated with.

A very similar principle can be found when looking at the training of embeddings for sentences and dialogues [Serban et al., 2016b]. The objective function for those also favors embeddings that allow a model to better predict the next words in a sequence. The models, however, are more complex, as the individual word embeddings of a conversation are combined in a hierarchical manner to first build embeddings on a response- and finally on a dialogue-level.

How well embeddings can be used to represent the context of a conversation depends on their quality: the semantic information that is encoded. This makes the training process and the corpus used for the training crucial.

1.1 Problem statement

Given a context, the dialogue that has taken place between two entities, finding a response that keeps the conversation going or leads to the solution of issues that might have been discussed is a non-trivial problem.

Exploring techniques that build on embeddings to give suitable responses and estimating the quality of embeddings is the goal of this thesis. Ultimately, a method building on embeddings acts also as an evaluator for them, as its performance is strongly influenced by the embeddings quality. This allows comparison and evaluation of different methods and hyper-parameters for the generation of embeddings. However, such estimations have to be analyzed carefully and critically. As indicated by the work of Pietquin [Pietquin and Hastie, 2013], evaluation of dialogue systems is an open problem. Currently, only through human evaluation, one can reliably measure conversation quality [Serban et al., 2016b, Serban et al., 2016c].

Models such as the Hierarchical Recurrent Encoder Decoder (HRED) [Sordoni et al., 2015], use a decoder to generate a response from a context embedding that was previously produced by its encoder component. A probability distribution over possible words from the decoder and a search technique such as beam search assemble a response word-by-word. This generative approach is very sensitive to the quality of embeddings, which can be seen in the difference between the complexity of answers generated with HRED and its improved version, the Variational Hierarchical Encoder Decoder (VHRED) [Serban et al., 2016c].

With a sufficiently large corpus, a suitable response might already be present and may also be grammatically acceptable, as it was produced by a human. Given a context embedding, a retrieval based approach could find acceptable responses, while being less dependent on a superior embeddings quality. However, the downside is that there are questions that can not be answered if the answer is not present in the corpus. Alternatively, the answer might not be entirely suitable.

Research questions

Given a retrieval based method such as Nearest Neighbor Search (NNS), through querying the context embedding of a conversation, dialogues and responses with similar meanings can be returned from the corpus. Based on this approach the first two research questions follow as:

- *How well do the dialogues returned by NNS fit to the context of the conversation at hand and how is this affected by the quality of embeddings?*
- *As the NNS returns multiple nearest neighbors, how can those be scored such that the best response can be found?*

Pre-trained word embeddings are available in many languages, trained on various corpora with differing size and context and usually serve as the basis for response and context embeddings. Their objective is to encapsulate as much global domain knowledge as possible, achieved by training them on corpora containing many different topics. An alternative is to train word embeddings on an available corpus of human-to-human conversations, generating embeddings with a stronger focus on specific domain knowledge. Choosing only one of the two means to sacrifice the other and, consequently, the third research question follows:

- *How can specific and general domain knowledge be encoded into an embedding and how does this affect the performance of models that build on them?*

1.2 Approach

The proposed solution to the aforementioned problem, how to respond given a certain context, is split into two parts: the encoding of response- and dialogue-meaning

into vector representations and the search for similar conversations by using an Approximate Nearest Neighbor (ANN) model.

The first step, generating vector representations of meaning, is performed by the HRED model, which is a Neural Network (NN) that consists of two Recurrent Neural Networks (RNNs), stacked in a hierarchical manner. The first RNN encodes a sequence of word embeddings to a response embedding at time step t. The context encoder, which is the second RNN, receives multiple response embeddings to generate a context embedding that represents the entire conversation up to time step t. In this manner, for each turn in a conversation, a response and context embedding is generated that is used within the training function and to represent existing dialogues.

The decoder component that is designed to generate a response, given the context embedding previously generated, is replaced by an ANN model. This is an instance-based approach, using distance metrics to find previously encoded dialogues with similar meaning. Nearest Neighbor approaches usually generate a speed-up structure over the existing data to find close neighbors more quickly and to avoid the problems of simple brute force search.

During the experiments, it became apparent that the closest neighbor does not necessarily contain the best answer, or even a good answer, often due to human error. An example for a very frequent mistake is the agent asking for the problem of the customer, even if it was already stated. A reason for this could be that agents start a conversation using a generic phrase and then start reading the customers response. However, for many cases, the best answer was amongst the k nearest neighbors. The last step of the answer-retrieval is the scoring of neighbors, which takes not only the question-to-question similarity into consideration but also the answer relevance. An overview of the whole pipeline can be seen in Figure 1.1.

1.3 Thesis structure

The thesis is structured into multiple chapters that range from introducing necessary basic concepts to reporting conceptual experiments and detailing specific implementations:

- **Chapter 2: Related work**

  In this chapter, basic concepts that are necessary to understand the hierarchical recurrent network approach that will be used to encode dialogues into embeddings, are discussed. However, the reader will first be introduced to dialogue systems and neural networks to ensure a basic understanding of the domain in which this thesis is placed in. Also, nearest neighbor search and ANN models will be discussed.

- **Chapter 3: Model Description**

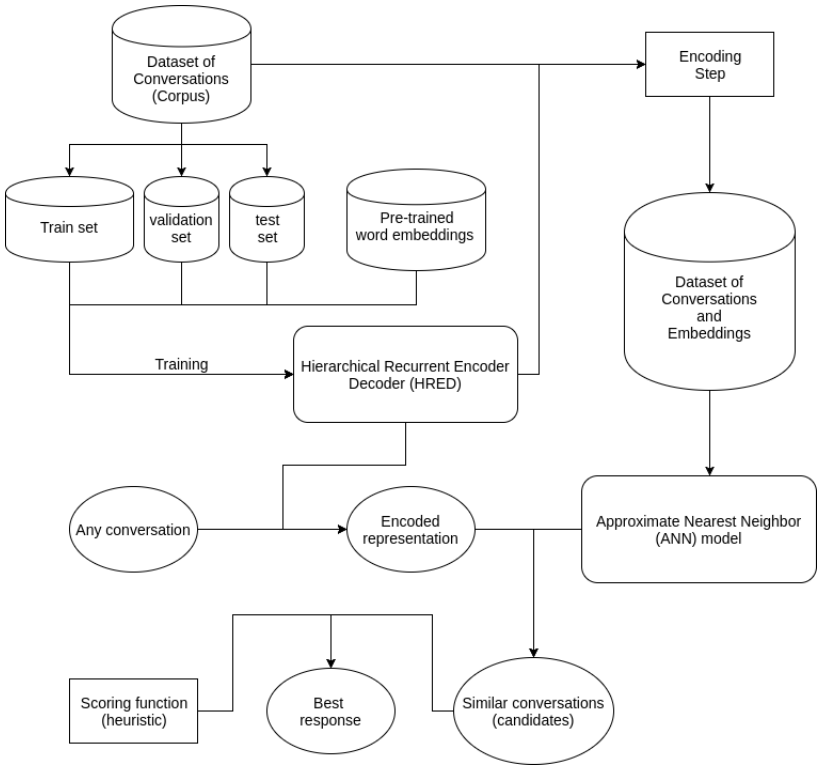

Figure 1.1: A view of the pipeline implementing the proposed approach. An HRED encoder is used to generate context and response embeddings and an ANN model builds on previous steps to propose possible responses.

First, given the previous introduction, the HRED model is explained in more detail. Secondly, the process of answer-retrieval, using an ANN model, and the scoring of neighbors is discussed in this chapter.

- **Chapter 4: Corpora and experiments**

  A human-to-human dialogue corpus (namely, the Vodafone's online customer service corpus) will be discussed in terms of its structure and the pre-processing that was performed to remove impurities and reduce the vocabulary size. The experiments will measure the performance of the proposed model by the Recall@k [Lowe et al., 2015] measurement, a method that is designed to measure ranking performance.

- **Chapter 5: Future work and final discussion**

  In the last chapter, improvements and extensions that target the model's

shortcomings or might form an interesting future research topic, will be suggested. The final discussion will summarize the results and the conceptual feasibility of the proposed approach.

Chapter 2

Related Work

This chapter will address the basics necessary for understanding how embeddings on a sentence- and dialogue-level can be generated. The first section will discuss Dialogue Systems and describe the reasoning behind the proposed retrieval based approach.

2.1 Dialogue Systems

The purpose of Dialogue Systems (DS), often also termed Conversational Agents (CA) or Chatterbots, is to converse with humans to provide information, help in decision making, perform administrative services, or just for the sake of entertainment [Shawar and Atwell, 2007]. The traditional design of Dialogue Systems [Jurafsky and James, 2000] follows a modular approach (see Figure 2.1), splitting the system usually into a *Natural Language Understanding* (NLU) module, a *Dialogue Manager* and a *Natural Language Generation* (NLG) unit. The NLU module processes the raw user input and extracts useful information and features that can be used by the Dialogue Manager to update internal states, send queries to a knowledge base or, more generally, find actions based on a script. The NLG acts inversely to the NLU module, receiving features and information from the Dialogue Manager to generate a response that, finally, will be presented to the user.

One of the simplest design approaches for an NLU is to simply spot certain keywords or combinations of them. This is often the general procedure of script-based chatbots and the approach followed by ELIZA. However, there is a long history of attempts [Bates, 1995] to improve NLU and find better representations of text. With advances in machine learning, the development ranges from statistical modelling of language [Manning and Schütze, 1999], semantic parsing [Dowding et al., 1993], skip-gram models [Mikolov et al., 2013b], and others, to approaches utilizing deep neural architectures [Collobert and Weston, 2008, LeCun et al., 2015]. Neural networks have also been used to improve NLG [Vinyals and Le, 2015, Sutskever et al., 2014]. Recently, Dialogue Managers have made similar advances towards au-

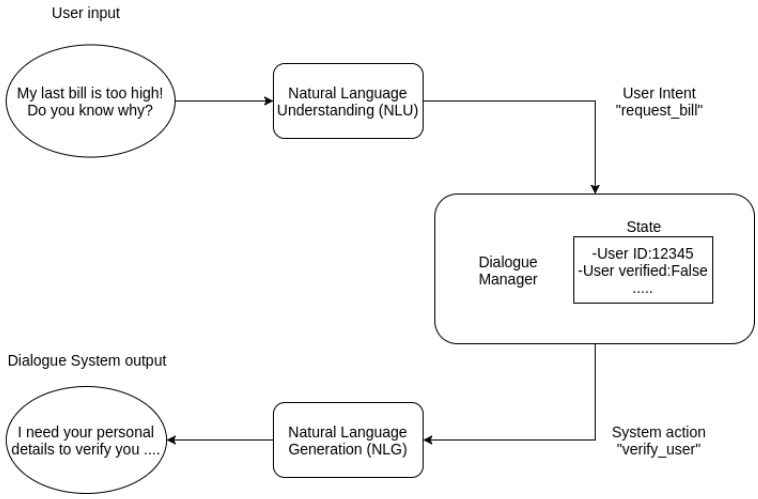

Figure 2.1: Schematic pipeline overview of a Dialogue System. Given the users input, the NLU module extracts intentions and features and forwards this information to the Dialogue Manager, which updates its internal state and finds suitable actions. The NLG module is responsible for rendering a response that formulates the Dialogue System's intention.

tomated solutions, with a focus on reinforcement learning [Shah et al., 2016, Young et al., 2013], generating policies of how to interact with humans, based on some state representation.

Modular or script-based Dialogue Systems (see Figure 2.2) are usually deployed by major companies that can carry large development costs and employ a sufficient number of developers and personnel. Writing a script, a list of rules of how to respond given the current state of a conversation can become difficult and extremely tedious, as one tries to cover most use cases of the company's services and possible user inputs. The development process is usually iterative, i.e, over multiple iterations, shortcomings are analyzed and the script of a Dialogue System is extended or altered. Besides large development costs, such systems are often fragile and difficult to maintain.

Data-Driven Dialogue Systems

With the rise of Deep Learning (DL) in recent years [Schmidhuber, 2015] and an increasing company interest in chatterbots, end-to-end Dialogue Systems, such as deep Recurrent Neural Networks, constituting all modules in one model [Serban et al., 2016b] (see Figure 2.3), have become one of the major research topics for Dialogue Systems. The tasks of NLU, NLG, and the Dialogue Manager are performed by a single deep network that is trained to reproduce conversations from a

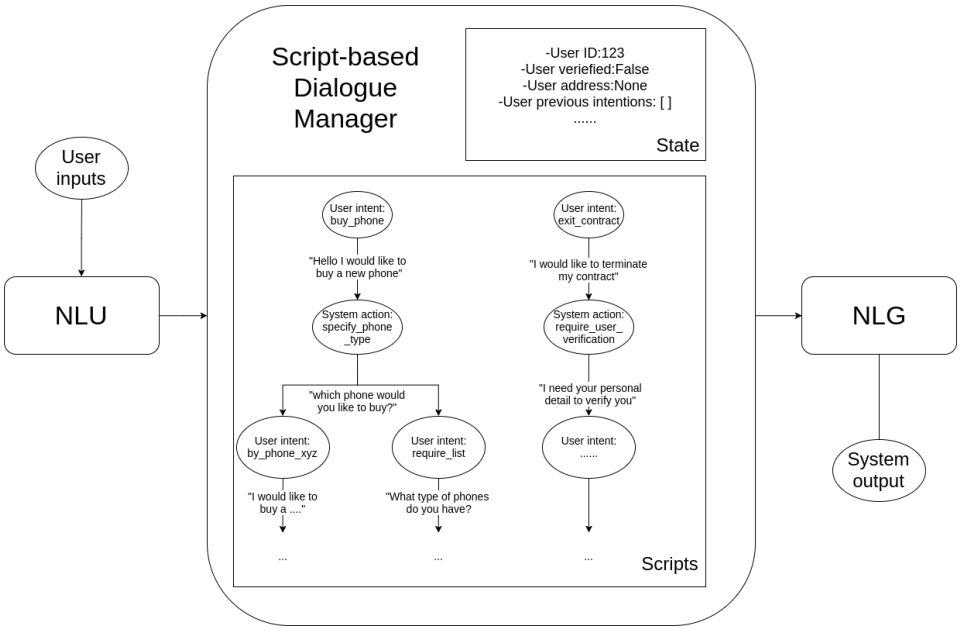

Figure 2.2: Example of a script based Dialogue Manager. The systems behaviour is essentially defined by a script that maps intentions to actions.

large dataset. Such a system would generate an answer end-to-end from raw user input. Even though training deep RNNs can be considerably difficult [Bengio et al., 2013], only one model would need to be optimized, and one could benefit from the neural model's capability to generate natural responses [Wen et al., 2015, Vinyals and Le, 2015]. The previously described development issues could also be tackled to a certain extent. However, such systems come with problems of their own, such as the difficulties of training and designing large and complex networks, the need for substantial amounts of human-to-human conversations, and the missing customizability of the system's behaviour, which can be of importance for companies that want to have more control over their dialogue system. The fact that all modules are represented by one model makes it also intrinsically difficult to design bots for administrative tasks, as a deep network is, amongst other things, incapable of authenticating a user on the base of reliable database queries, or of sending information requests itself. To implement such a behaviour, it would be necessary to annotate the corpus with meta information about customer service actions and provide hidden customer information that is stored in a separate database. The task of reliably annotating the corpus, possibly finding a vector representation for customer data and adapting an end-to-end Dialogue System to the new input would exceed the scope of this thesis.

Data-driven Dialogue Systems are characterized by their strong dependency on

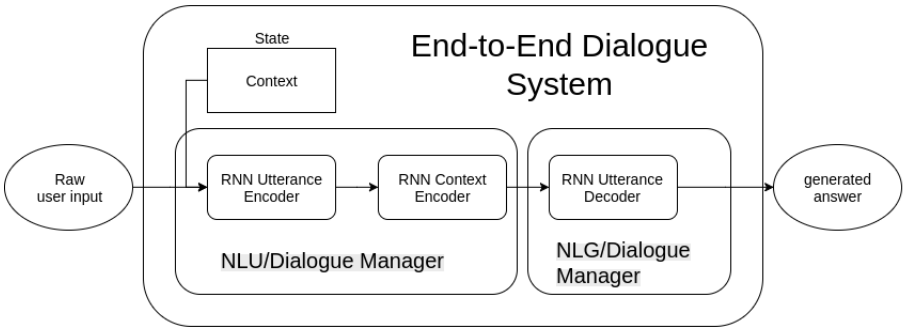

Figure 2.3: This figure shows how regular Dialogue Systems relate to end-to-end Dialogue Systems by visualizing the individual HRED components and their tasks.

large corpora of human-to-human conversations to implement models that imitate human behaviour. The main distinction of such systems can be made between generative- and retrieval-based approaches. A retrieval based model [Banchs and Li, 2012] would search a database for suitable answers and score them based on a heuristic function, while a generative approach generates a response word-by-word, optimizing the likelihood of consecutive words given the previous context of a conversation. Both approaches rely on the quality and amount of conversations when retrieving possible answers or providing versatile training examples to a deep network.

Large datasets of human-to-human conversations can contain sensitive personal information, especially considering online customer service chats. Legal concerns and the availability of such datasets [Serban et al., 2015] restricts the usage of deep networks for Dialogue Systems mostly to large companies that maintain their own online customer service and collected sufficient amounts of conversations over the years. With the intrinsic Dialogue Manager of a deep network suffering from a lack of customizability, access to the outside world, and missing insights to its behaviour, the question remains of how to make end-to-end Dialogue Systems applicable for large companies.

The VHRED model, an advanced version of the HRED model, was originally trained on the ubuntu corpus [Lowe et al., 2015], a dataset containing human-to-human conversations of ubuntu-related problems. With end-to-end models fundamentally depending on the data they were trained on, a model trained on the ubuntu corpus would, due to the structure of the data, mostly perform information provision, rather than administration. Such models could be useful for large companies that want to provide quick answers to frequently asked questions. For the training of end-to-end models, one might consider using the human-to-human conversations from an online customer service. However, a problem arises from the following fact: a customer service agent performs both tasks, information provision

and administration. The agent can also have access to personal information of a customer that cannot be inferred from the context of a conversation. As such, the resulting corpus is filled with general questions that can be answered and personal questions that would require additional information to be answered correctly. This mixture makes it difficult to train a model that only focuses on frequently asked questions. Also, with this setup, performing customer-specific services is impossible.

However, end-to-end models generate rich context embeddings that could be used as a state representation or as an addition to existing representations. In this thesis we will explore how a retrieval-based approach can utilize the embeddings generated by an end-to-end model to improve its performance. This combination is also compatible with the traditional modular design. Before specifying the proposed model in the next chapter, the basics for generating embeddings with neural networks will be introduced in the next sections.

2.2 Neural Networks

Neural Networks (NNs) are an attempt to mimic the problem-solving behaviour of the human brain and apply it to machine learning tasks [Lopez et al., 2008]. Similar to the biological counterpart, Neural Networks are constituted of connected neurons that amplify or dampen signals sent between each other. Limited by the hardware architecture of computer processors, memory storage and the necessary insights into the human brain, current Neural Networks are just abstract models of the concepts one knows of the brain. However, they have proven to be useful in several machine learning domains such as image classification [Krizhevsky et al., 2012], speech recognition [Hinton et al., 2012], computing distributed representations of text [Serban et al., 2016b, Serban et al., 2016c], state value approximation [Silver et al., 2016] and many others.

A typical NN consists of several connected layers of neurons where each layer receives an input signal from the previous layer, computes an output signal and passes it to the next layer. Before forwarding, an activation function is applied that squashes the signal. Such functions are typically non-linear and prevent the sum of signals from exceeding or falling below certain thresholds. Besides the structure and activation function of a NN, it is also characterized by the strength of its connections, called weights. As signals are sent between neurons, the weights are responsible of how they change. Through a training process [Bottou, 2012, Lopez et al., 2008], which is guided by an objective function, weights are tuned in such a way that the network produces desirable output values given certain input values. As weights are the main modifiable characteristic of a network, they are also sometimes referred to as the program or memory of the network.

2.2.1 The perceptron neuron model

For the most types of neural networks, individual neurons are modelled with the perceptron scheme (see Figure 2.4), a simplified mathematical representation of the biological neuron.

A perceptron receives input values $x_1, ..., x_n$, each weighted by $w_1, ..., w_n$ and the input from a bias unit b to calculate *net*, the entire input to the neuron. In the perceptron model, the combination of input values follows following equation:

$$net = b + \sum_{i=1}^{n} x_i * w_i \tag{2.1}$$

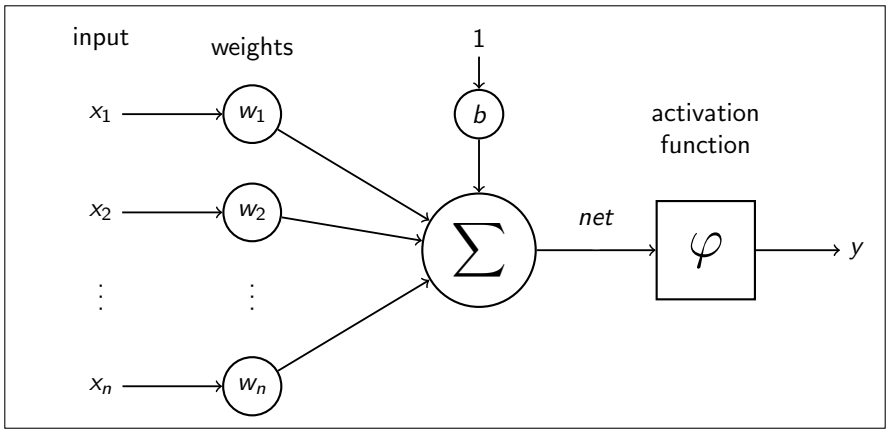

Figure 2.4: Model description of a single neuron.

To restrict the accumulated value to a certain range, an activation function φ is applied, which is usually the sigmoid or the hyperbolic tangent function. The result of the activation function y is the final output of the neuron.

The hyperbolic tangent function

$$\varphi(net) = tanh(net), \tag{2.2}$$

squashes the input *net* and restricts the outcome to be in the range of $[0, ..., 1]$.

The sigmoid function

$$\varphi(net) = \frac{1}{1 + e^{-net}}, \tag{2.3}$$

binds the outcome to $[-1, ..., 1]$.

Considering the fact that the input values to a neural network have a certain range and mean, one needs to take into account that the activation function can change

these as the values progress through the network. For instance, if the input values contain negative numbers, a sigmoid function would bind every negative value to $[0, 0.5]$, shifting the mean from 0 to 0.5. This does not mean that the network would not be able to produce any useful results, but usually, an activation function that fits the criteria of the input values will perform better. Often such design choices are overlooked because of networks performing well enough.

2.2.2 Feedforward Neural Networks

The main characteristic of feedforward NNs [Lopez et al., 2008], also called Multilayer Perceptrons (MLPs), is that they contain no cycles. This means that signals sent through the network will be encountered only once in a neuron. Such networks (see Figure 2.5) are modelled through consecutive layers that pass their outcome only to their successor. This allows speeding up computations by using efficient matrix multiplications and, additionally, the designing process of networks is simplified.

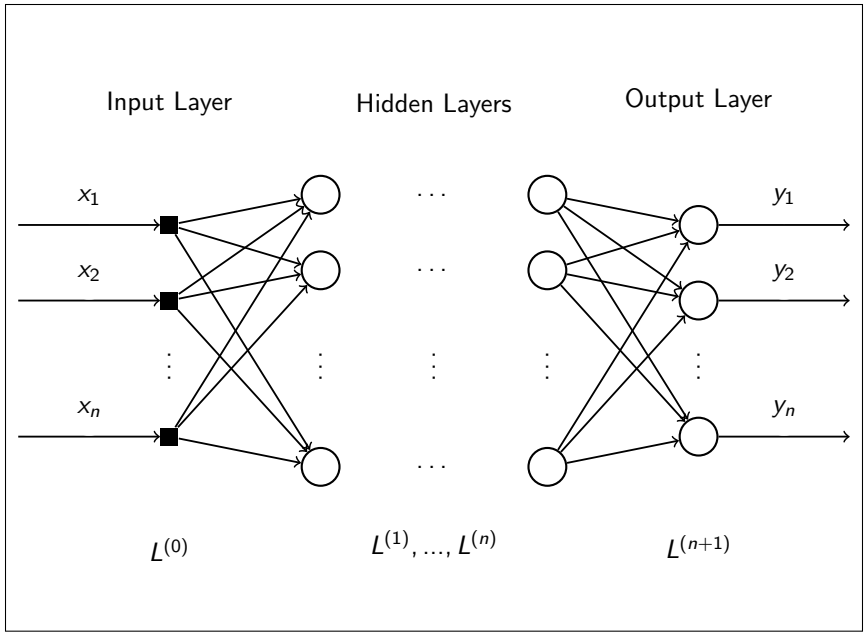

Figure 2.5: Multi Layer Perceptron (MLP) architecture.

The typical design convention is that neurons are grouped into L layers, with one input layer $L^{(0)}$, an arbitrary amount of hidden layers $L^{(1)}, ..., L^{(n)}$, and one output layer $L^{(n+1)}$, with n being the number of hidden layers and h_n number of neurons per layer. In general, a feedforward NN is fully connected, i.e., a neuron in layer $L^{(1)}$ is connected to each neuron in layer $L^{(2)}$. The input layer can be seen as a

set of placeholders, as its purpose is only to hold the network input x to pass it to the first hidden layer and no activation function is applied. The neurons in the hidden layer follow the convention of the perceptron scheme, applying an activation function after accumulating the results from the previous layer. However, activation functions may be different for certain layers, especially for the last. For the neurons in the output layer, one could choose a softmax function to generate a probability distribution and use it for sampling or classification tasks.

The structure of Feedforward Neural Networks allows the grouping of computations of individual neuron input and output values by using matrix multiplications. This approach allows the utilization of the large parallel computation capabilities of Graphic Processing Units (GPUs), or the less powerful vector computation instructions of the Central Computing Unit (CPU). An efficient and convenient way to represent weights between layer l and $l-1$ is to store them in a matrix $W^{(l)}$, with $W^{(l)}_{i,j}$ being the weight between neuron j in layer $l-1$ and neuron i in layer l. By using the dot product between weights and layer input, $net^{(l)}$ can be efficiently calculated as:

$$net^{(l)} = W^{(l)} \bullet x^{(l)} + b^{(l)}, \tag{2.4}$$

with $x^{(l)}$ being the output of the previous layer. The output $y^{(l)}$ of layer l is then calculated as:

$$y^{(l)} = \varphi(net^{(l)}), \tag{2.5}$$

using activation function φ.

The training and design of Neural Networks, especially deep variants, is a complex research domain in itself [Bengio, 2012], considering the many design-related choices one can make, the possible ways to pre-process the data and training procedures, which are hard to debug. Initializing weights properly, choosing activation functions with suitable non-linearity, training with an adapting learning rate, or monitoring weights during training are just a subset of possible considerations one can make to adapt a network to the problem at hand.

Using the usual training method, Stochastic Gradient Descent (SGD) [Bottou, 2012] with Backpropagation, an error is calculated based on the networks prediction $\hat{y}_t$ to a training instance (x_t, y_t), with y_t being the desired prediction. By backpropagating the error layer-wise through the network and calculating a gradient, a weight update to decrease the error, one can train such a network to predict labels $\hat{y}_t$ that are close to desired labels y_t. Gradients are based on the error of previous neurons, layer input $net^{(l)}_t$, and the derivative of activation functions. If one chooses a sigmoid function, or an activation function that squashes a signal in general, derivatives tend to become very small, usually in the range of [0..1]. When propagating an error through multiple layers, it becomes smaller at each step, which can finally lead to almost zero-sized gradients at the earliest layers and, therefore,

to poor training results. This effect is also referred to as the vanishing gradient problem, usually occurring in combination with deep neural architectures.

2.2.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are similar to Feedforward networks, as both are fully connected and posses layered structures. However, by connecting neurons of the same hidden layer with each other, thereby creating cycles, the results of an RNN become time dependent, i.e., intermediate results at time step $t-1$ are influencing the outcome at time step t (See Figure 2.6). This recurrence allows the network to roughly remember previously seen input values. Thereby it can process arbitrary long input sequences, a form of representation that is incompatible with fixed-length-input dependent Feedforward networks. Considering the time aspect, the computation of the network input $net^{(l)}$ for hidden layer l changes to

$$net_t^{(l)} = W^{(l)} \bullet x_t^{(l)} + R^{(l)} \bullet y_{t-1}^{(l)} + b^{(l)}, \tag{2.6}$$

using recurrent connections $R^{(l)}$ and $y_{t-1}^{(l)}$, the output of hidden layer l at previous time step $t-1$. The computations for the network output stay the same:

$$y_t^{(l)} = \varphi(net_t^{(l)}). \tag{2.7}$$

The output $net_t^{(l)}$ is often related to as the hidden state, context state or the memory of the network, as it summarizes current and previous inputs.

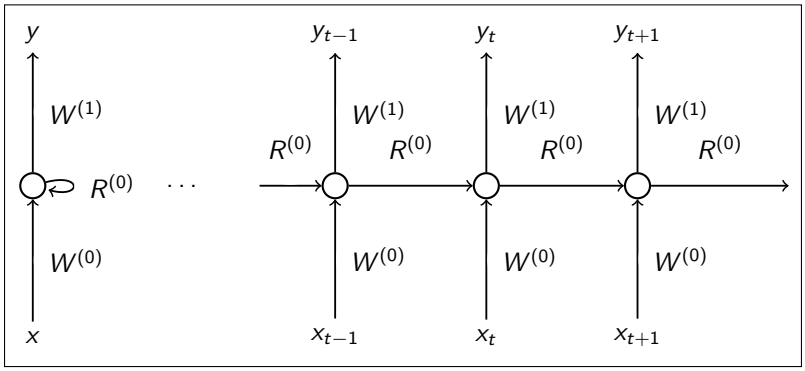

Figure 2.6: Example of an RNN unfolded in time (right) and its original version (left). The network computes activation values for its hidden layer using weights $W^{(0)}$ and recurrent weights R.

RNNs are useful to process textual data, with sentences being usually represented as sequences of word embeddings. Such a sequence can be step-wise encoded into a hidden state, and when passed as input to a softmax function, used to calculate a probability distribution over possible words in a dictionary. Using such distributions,

one can predict the next words in a sequence and also generate whole sentences. Additionally, the hidden state, as it is a summary of previous words, can be used as a new representation form of the processed sentence (also called utterance, response, or sentence embedding).

What words are predicted and the quality of those predictions may depend on how the network was trained, its initialization, the number of hidden units, and the design choices of the network in general. Choosing too few hidden neurons could result in the network having difficulties to keep track of longer context dependencies and in less expressive sentence embeddings. Choosing too many neurons leads to an increase in the necessary time for training such a network. Also, the network could suffer from so-called 'overfitting' [Bengio, 2012], a problem regarding the training where the network learns to accurately reproduce training examples instead of becoming able to generalize over them, either due to too much training, faulty training, or the size of the network. Such networks usually perform poorly when encountering unseen data. Design schemes for modeling the hidden state, the Short-Term Memory (LSTM) [Hochreiter et al., 1997] and the Gated Recurrent Unit (GRU) [Cho et al., 2014], have been proposed to overcome training related difficulties and to improve an RNN's capability to keep track of long-term dependencies.

2.2.4 Long Short-Term Memory

One of the main problems of plain RNN's is that proper training is fundamentally difficult [Bengio et al., 1994]. When applying Backpropagation Through Time (BPTT) [Williams and Zipser, 1989] to train a RNN, gradients are calculated based on a chain rule that takes previous time steps into consideration. The problem is that gradients, calculated for a certain time step, become consecutively smaller, the larger the temporal difference is. This issue relates to the previously discussed vanishing gradients problem, a problem which mostly affects deep architectures. A RNN with a single hidden layer may not be considered a static deep network, but unfolded in time, similarities become apparent. As a deep network would suffer from poor accuracy, a RNN would have difficulties in learning long-term dependencies.

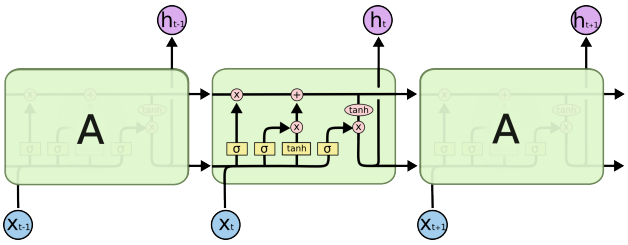

Figure 2.7: Visualization of a Long Short-Term Memory cell. Figure originates from [Olah, 2015].

LSTMs, first introduced 1997, are an attempt to increase a RNN's capabilities to

remember those dependencies by replacing hidden units with more complex memory cells (see Figure 2.7), capable of controlling the information flow in and out of the cells. This control is usually modelled through three gates, the input gate i_t, the forget gate f_t and the output gate o_t, all operating on a cell state c_t and conditioned on the current input x_t and previous result y_{t-1}. They are parametrized by W, U and b, such that forgetting and remembering can be influenced through training. The gate output values are calculated as:

$$\begin{aligned} f_t &= \varphi_g(W_f x_t + U_f y_{t-1} + b_f) \\ i_t &= \varphi_g(W_i x_t + U_i y_{t-1} + b_i) \\ o_t &= \varphi_g(W_o x_t + U_o y_{t-1} + b_o), \end{aligned} \tag{2.8}$$

with φ_g being the sigmoid function. How much is kept of the previous cell state c_{t-1} is controlled by entry-wise multiplication with the vector f_t, the forget gate's result. The input gate i_t controls how much of the current input will affect the final cell state. Using the forget and input gate, the cell state c_t of time step t is calculated as:

$$c_t = f_t \circ c_{t-1} + i_t \circ \varphi_c(W_c x_t + b_c), \tag{2.9}$$

with φ_c being the hyperbolic tangent function and W_c and b_c the cell state's parameters. How much of the cell state is passed to the rest of the network is controlled by the output gate:

$$h_t = o_t \circ \varphi_o(c_t), \tag{2.10}$$

with φ_o usually being the hyperbolic tangent function. However, there are many variants of the original LSTM approach. For instance, a "peephole" version [Gers and Schmidhuber, 2000] does not use the second activation function φ_o and proposes to condition the gates also on the cell state c_t, using peephole connections between gates and cell state.

2.2.5 Gated Recurrent Unit

The more recently proposed GRUs (see Figure 2.8) are similar to LSTM cells aiming to improve a RNN's capabilities to remember long-term dependencies. However, they use a different gate design, have fewer parameters to train and are without an additional cell state. Two gates, the reset and update gates r_t and z_t, operate directly on the hidden state, i.e., the hidden layer. Parametrized by W, U and b, while conditioned on the current input x_t and previous result y_{t-1}, GRU gate vectors are computed similarly to LSTM gates:

$$\begin{aligned} z_t &= \varphi_g(W_z x_t + U_z y_{t-1} + b_z) \\ r_t &= \varphi_g(W_r x_t + U_r y_{t-1} + b_r), \end{aligned} \tag{2.11}$$

with φ_g being the sigmoid function. The update gate z_t combines the function of the input and forget gate by controlling how much the new hidden state is defined by either the current input or the last hidden state, using linear interpolation:

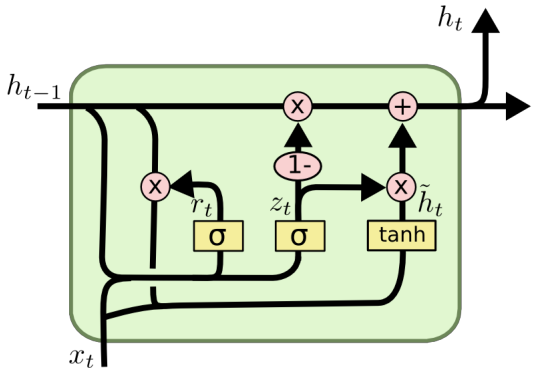

Figure 2.8: Visualization of a Gated Recurrent Unit (GRU). Figure originates from [Olah, 2015].

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t, \tag{2.12}$$

with $\tilde{h}_t$ being the candidate activation. The reset gate r_t is used to calculate $\tilde{h}_t$, controlling similarly how much of the previous hidden state to keep:

$$\tilde{h}_t = \varphi_h(W_h x_t + U_h(r_t \circ h_{t-1} + b_h), \tag{2.13}$$

with φ_h being the hyperbolic tangent function.

The LSTM and GRU approaches and, often, also variations of them, have been compared [Chung et al., 2014, Jozefowicz et al., 2015], and the results show that there has not been found a superior design that consistently outperforms the others. As all gated designs perform better than the basic RNN architecture and are more or less equally powerful, one just might choose a basic gate design and focus on other hyperparameters.

2.3 Summary

In the first section, Dialogue Systems and their different types have been discussed. The difficulties in training end-to-end Dialogue Systems on corpora of customer service conversations lead to the conclusion that using a traditional modular approach in combination with rich context embeddings might be better suited for large companies that want to have more control over their chatterbot. The model that is proposed in this thesis (see Figure 1.1), a combination of retrieval-based approach and rich embeddings, will be constituted of three parts: the HRED model that encodes text, a nearest neighbor approach that searches for possible answers, and a scoring module that evaluates answers. The first part, the HRED model, is essentially a deep recurrent network that stacks GRU-based RNNs, which were discussed in the previous section. Feeding the conversations of a corpus to the HRED

model, the hidden states of these networks, the embeddings, will act as a different form of representation, creating a new database on which nearest neighbor search can be applied on.

Chapter 3

Model Description

This chapter will describe the proposed pipeline in more detail. First, the HRED model will be discussed and how it utilizes RNNs. The second part will cover nearest neighbor search (NNS) and approximate nearest neighbor search, a variation that is better suited for high-dimensional data. The final part will discuss scoring methods for the responses retrieved by NNS and how the individual parts come together.

3.1 Hierarchical Recurrent Encoder-Decoder

Textual conversations have the intrinsic property of people interacting with each other, usually two or more. As such, they differ from common plain text. Regarding objective functions for training, the property of multiple speakers or writers can be very useful for predicting the next words in a sequence, considering that the response of one person strongly depends on the context of the conversation.

The Hierarchical Recurrent Encoder Decoder (HRED) model, first introduced by [Sordoni et al., 2015] to predict search queries, was extended by [Serban et al., 2016b] to generate response and context embeddings to represent entire dialogues and the turns the dialogues are constituted of. The HRED model, designed to represent NLU, dialogue manager and NLG of a dialogue system, allows through its hierarchical design to consider speaker turns and conditions the generation of responses on the entire context of the previous conversation. This makes the HRED model superior to RNN language models [Mikolov et al., 2010] that condition response generation only on the previous utterance.

A corpus consisting of human-to-human conversations in a customer service domain, the online Vodafone chat service, is the basis for this thesis. The complete and anonymized corpus was processed by the HRED model to generate embeddings for individual responses and entire dialogues, with the HRED model being trained previously on the corpus. Dialogues and embeddings are saved to a database for further usage.

As the corpus consists of dialogues in Dutch, pre-trained word embeddings from [Tulkens et al., 2016] have been used to initialize a vocabulary. Words for which no pre-trained embedding has been found received random word embeddings.

3.1.1 Architecture

In this section, the architecture of the HRED model will be described as it was first introduced in [Sordoni et al., 2015] and used in this thesis to compute context and utterance embeddings.

The HRED model (see Figure 3.1) essentially consists of three stacked RNNs: the utterance encoder, context encoder, and utterance decoder, each of them depending on the result of its predecessor and operating on distributed representations.

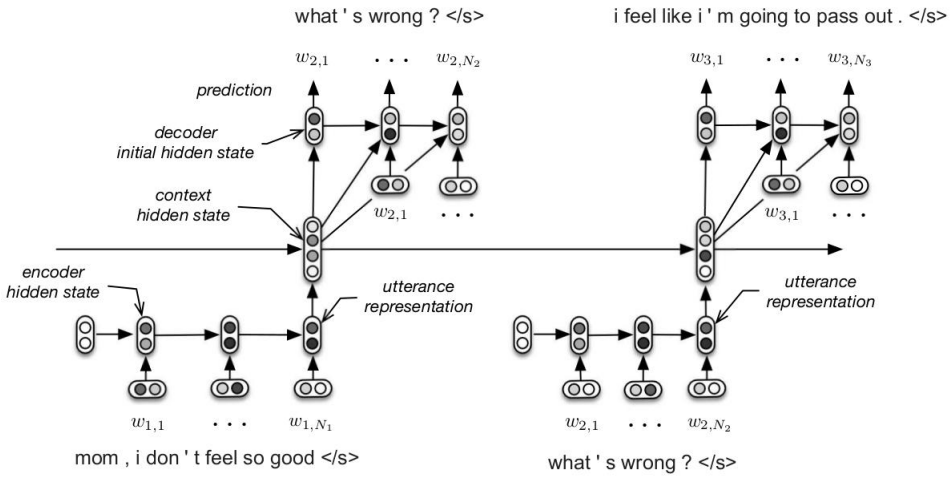

Figure 3.1: This figure shows the HRED pipeline over multiple conversational turns. The utterance encoder processes the input sentence *"mom, i don't feel so good </s>"* to generate its hidden state, the utterance representation. The context encoder receives this embedding together with the context embedding of the previous turn to compute the current turn's context hidden state. By initializing the decoder's hidden state with this context embedding, the response *"what's wrong ? </s>"* is generated word-by-word. The generated response serves as input to the utterance encoder in the next turn. Taken from [Serban et al., 2016b].

Formally, a dialogue D, the input to such a model, can be represented as a sequence of utterances $D = (U_1, ..., U_M)$, with U_m being a sequence of word indices $U_m = (w_{m,1}, ..., w_{m,N_m})$, each of them usually pointing to a vocabulary reference or directly to a word embedding. These become the input to the utterance encoder.

Encoding steps

To better capture long-term dependencies, the GRU gating function is used for the individual RNNs. For a simplified notation, the GRU can be expressed compactly by combining equations 2.11, 2.12 and 2.13:

$$h_t = GRU(h_{t-1}, x_t), \tag{3.1}$$

computing current hidden state h_t, conditioned on the previous, h_{t-1} and on current input x_t. To comply with the HRED notation, the utterance embedding $h_{m,n}$ of the current utterance U_m, including word $w_{m,n}$, is calculated as:

$$h_{m,n} = GRU_{utt}(h_{m,n-1}, w_{m,n}). \tag{3.2}$$

Applying equation 3.2 consecutively on word embeddings $w_{m,1}, ..., w_{m,N_m}$, results in an equally-sized set of hidden states $h_{m,1}, ..., h_{m,N_m}$, where the last hidden state h_{m,N_m} is the summary of all words in the same utterance. As such, we denote $h_m = h_{m,N_m}$ to be the hidden state that represents utterance U_m.

Using this encoding approach, a set of utterances $U_1, ..., U_M$ is encoded into hidden states $h_1, ..., h_M$. Those are used as input to the GRU-based context encoder, similar to how word embeddings acted as input to the utterance encoder. As such, context embeddings c_m are a summary of utterances and represent entire dialogues. They are computed as:

$$c_m = GRU_{con}(c_{m,n-1}, h_m). \tag{3.3}$$

3.1.2 Decoding step

In addition to encoding a sequence of embeddings into a hidden state, the decoder component generates word probabilities over a vocabulary, given some context $U_1, ..., U_{m-1}$ and previous words $w_{m,1}, ..., w_{m,n-1}$.

Firstly, to condition the decoder RNN on previous utterances, the initialization of its hidden state is based on the context encoders last hidden state c_{m-1}. If not designed explicitly, context and decoder RNN usually have different hidden state dimensionalities, which is why an additional network layer is added to project context embeddings into the decoder space:

$$d_{m,0} = tanh(D_0 c_{m-1} + b_0), \tag{3.4}$$

with parameters D_0 and b_0 and $d_{m,0}$ being the decoder RNN's initial hidden state.

Given a set of words $w_{m,1}, ..., w_{m,n-1}$, having been previously generated or representing a training example, the decoder RNN hidden state is similarly computed as it was done for the encoder RNNs:

$$d_{m,n} = GRU_{dec}(d_{m,n-1}, w_{m,n}), \tag{3.5}$$

processing words consecutively. The first iteration uses the hidden state computed by equation 3.4 and a zero-value embedding for $w_{m,0}$ to predict the first word of an utterance.

Using both, the hidden state $d_{m,n-1}$ and word embedding of $w_{m,n-1}$, the word embedding of current word $w_{m,n}$ is then predicted as:

$$w(d_{m,n-1}, w_{m,n-1}) = H_0 d_{m,n-1} + E_0 w_{m,n-1} + b_0, \tag{3.6}$$

with the additional parameters H_0, E_0 and b_0. H_0 and E_0 control which part of the previous context- and word embedding contribute to the new word embedding and how much of that part is used.

By calculating the dot product of such generated word embeddings with the embeddings in a vocabulary, one can compute the similarity between prediction and existing words, with the most similar word being the most likely one. The actual probability of a word occurring next is based on this similarity:

$$P(w_{m,n} = v | w_{m,1:n-1}, U_{1:m-1}) = \frac{exp(e_v^\top w(d_{m,n-1}, w_{m,n-1}))}{\sum_{k=1}^{K} exp(e_k^\top w(d_{m,n-1}, w_{m,n-1}))}, \tag{3.7}$$

with e_v being the word embedding of word v, w the predicted word embedding and K the vocabulary size.

By computing probabilities for each word in the vocabulary, one can create a distribution from which words can be sampled. Pushing sampled words back into the decoder allows to generate the next word, extending an utterance until an end-of-sequence meta token has been reached. Possible generations can be explored using probability based search techniques such as Beam Search [Cho et al., 2014].

Given a training instance, a dialogue $D = U_1, ..., U_{m-1}$ of size M, one can score the HRED model based on how well it is capable to consecutively predict the words of each utterance $U_m = (w_{m,1}, ..., w_{m,N_m})$. Maximizing the log-likelihood of consecutive words, the training objective is defined as:

$$F(D) = \sum_{m=1}^{M} \sum_{n=1}^{N_m} log(P(w_{m,n} | w_{m,1:n-1}, U_{1:m-1})). \tag{3.8}$$

Summary

In this section, an end-to-end Dialogue System has been introduced. This system encodes the meaning of dialogues or the context of a conversation into a new

representation form, namely the context embeddings. The goal is to generate embeddings that encapsulate long-term dependencies, semantic information and other rich features. Natural and context-aware responses are expected when their generation is conditioned on these context embeddings.
The training occurs on a corpus of customer service dialogues, with the goal of reproducing such conversations. The final model is used to encode the original corpus, resulting in a database of context and utterance embeddings. This database will be used in the next step to find similar conversations, given some search query.

3.2 Nearest Neighbor Search

The term Nearest Neighbor Search (NNS) defines a class of search algorithms that operate on a dataset to find the closest or most similar match to a given search query (see Figure 3.2). Formally, a single data point p, contained in a data set P, is defined as a vector of features that lies in $\mathbb{R}^d$. The similarity between p and a query vector $q \in \mathbb{R}^d$ is computed by a distance function D. If one follows the naive approach, also called "brute-force" search, to find the closest neighbor, distances between search query q and all instances in P have to be computed, resulting in $\mathcal{O}(dn)$ linear complexity, with d being the dimensionality or feature length.

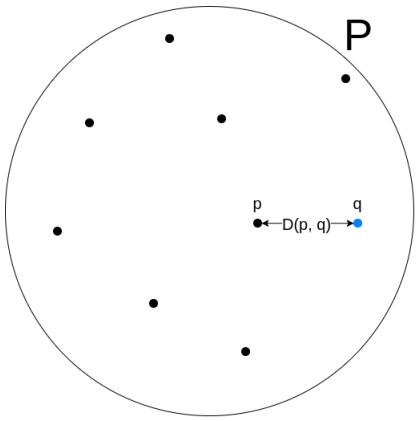

Figure 3.2: This figure shows a schematic overview of nearest neighbor search. The blue dot represents a query point q, which is linked to the closest neighbor p in the whole dataset P. The distance between two points is calculated by a distance function $D(.,.)$.

With computer storage capacities increasing quicker than computation power, it becomes more and more difficult to search the amounts of data that are accumulated nowadays (and in the future). Certain speed-up structures aim to decrease time-complexity desirably to $\mathcal{O}(\log n)$. This is often achieved by partitioning data points such that distance measurements do not have to be computed for all in-

stances in P, but for a considerably smaller subset.

An example for a popular speed-up structure is the k-dimensional Tree (k-d Tree). Each node in the tree, represented by a single data point, acts as a hyperplane that splits the data in half (see Figure 3.3). In the first tree-building iteration, a data point is chosen that would best separate the data according to the first dimension. The process continues with the left and right child each receiving half of the data and being linked to the second dimension. After reaching the last dimension, the data is split again according to the first dimension.

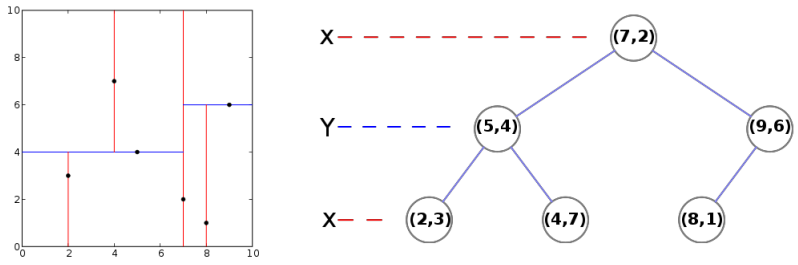

Figure 3.3: The image on the left shows how a data-set is portioned by a k-d Tree (right). Each node contains a single data point $p \in P$. Retrieved March 16, 2017 from Wikipedia: `https://en.wikipedia.org/w/index.php?title=K-d_tree&oldid=766561677`.

The problem with space-partitioning approaches is that they suffer from "the curse of dimensionality", a decrease of performance that occurs when dimensionality increases [Weber et al., 1998]. In the worst-case, such techniques can degrade to linear search, which makes space-partitioning methods unsuited for high-dimensional data.

Approximate nearest neighbor (ANN) approaches provide a better performance scaling as dimensionality increases. Instead of looking for the best match, ANN methods return sufficiently close neighbors. By loosening the restriction to find the best match, these techniques can reach sublinear complexity $\mathcal{O}(\log n)$.

An ANN implementation, provided by the python library scikit [Pedregosa et al., 2011], the locality-sensitive hashing forest (LSH-Forest), has been used in the proposed pipeline. In the following sections, basic locality-sensitive hashing (LSH) [Har-Peled et al., 2012] and the more advanced LSH-Forests [Bawa et al., 2005] will be introduced.

3.2.1 Local Sensitive Hashing

The basic idea of LSH is to project data points $p \in P$ into buckets that contain similar points. Hashing or projection functions, linked to each bucket, compute if a point belongs to a bucket (1) or not (0). Functions that maximize the probability that similar points are grouped into the same bucket (1) and lower the collision with dissimilar ones (0), are preferred. Formally, a hashing function h, belonging to a family of functions $\mathcal{H}$, is considered interesting, if, with a high probability P_1, similar points are grouped together:

$$\text{if } D(p,q) \leq r, \text{then } h(p) == h(q), \tag{3.9}$$

with r being a threshold that defines when two points are close. Furthermore, a hashing function should minimize probability P_2 of colliding dissimilar points:

$$\text{if } D(p,q) > r(\epsilon+1), \text{then } h(p) == h(q). \tag{3.10}$$

with ϵ being an additional approximation factor. A desirable family of functions $\mathcal{H}$ that ensures for each point pair p, q that $P_2 < P_1$ is called (r, ϵ, P_1, P_2)-sensitive [Har-Peled et al., 2012]. LSH can only work if there is a family of functions $\mathcal{H}$ that fits to the distance function D [Bawa et al., 2005] and as such, satisfies the aforementioned property.

The probability of undesired collisions can be further decreased by applying multiple hashing functions $h_1, h_2, ..., h_k$ to the same data point. The probability that data points get hashed multiple times into the wrong bucket decreases with the number of hashing functions: P_2^k. This allows lowering the probability of unwanted collisions to a sufficiently small value.

Furthermore, as one applies k hash functions on a single data point, a set of truth values or a binary string results, in which each entry states if the according hash function returned one or zero. This string or label can be used in a hash-table to quickly index a bucket, which contains the original data point.

By applying multiple hashing functions, not only P_2 decreases, but so does P_1, the probability of similar points being hashed into the same bucket. However, if we have chosen a suitable family $\mathcal{H}$, the ratio of P_1 and P_2 will still increase: $(P_1/P_2) < (P_1/P_2)^k$ [Slaney and Casey, 2008].

To solve the problem of possibly not finding any nearest neighbor, multiple hash-tables L are generated, each of them constructed with a different set of uniformly sampled hash functions.

Given a query point q, the first step of finding close neighbors in a set of l LSH-Tables is performed by computing l different labels for each individual hash-table. Using these labels as keys, buckets can be indexed in each table, given that there is a collision. These buckets can then be searched for the closest neighbors, using brute-force search and the original distance function D. Querying with a single

hash-table can be seen in Figure 3.4.

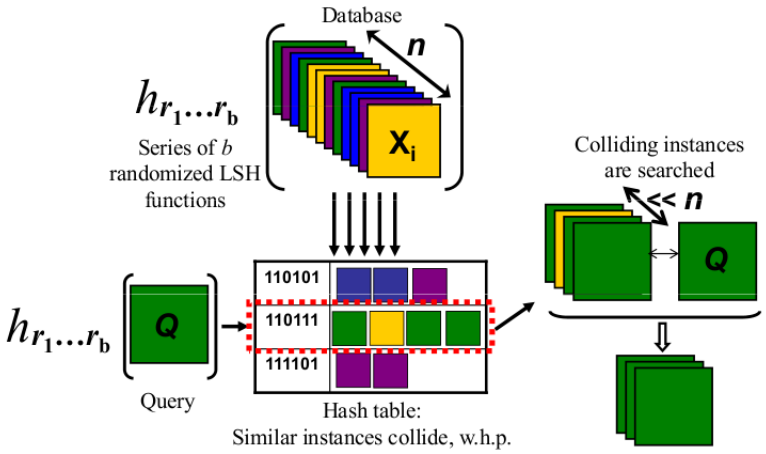

Figure 3.4: This figure shows the computational graph of querying with a single LSH-Table. The first step is to apply a set of hashing functions to generate the query's binary string label, which for this example is "110111". Using that label as a key in the shown hash-table results in a bucket containing four instances with the same label. The final step is to search those four data points for the best or k-closest matches with the query point. Creator of this image is Michael Vogiatzis (2013, September 8). Retrieved March 21, 2017 from the World Wide Web: `https://micvog.com/2013/09/08/storm-first-story-detection/`.

One of the main issues with the basic LSH algorithm [Bawa et al., 2005] is that choosing the optimal number of hashing functions k and number of tables l requires one to know the most suitable value for r, the threshold separating similar and dissimilar points. However, as there is often no perfect threshold that suits each query, a value that covers most cases is used.

LSH-Forests is a technique that addresses this issue and will be introduced in the next section.

3.2.2 LSH Forest

The main distinction between LSH Forests [Bawa et al., 2005] and the previously discussed algorithm is the way string labels are built and used to index data points or buckets. Instead of restricting the length of a binary string label to a fixed value k, labels are extended until they are unique, i.e., a data point is processed by as many hashing functions as necessary to ensure that the resulting binary string is unique amongst all other labels.

For the case that two points are very similar or equal, the length-limiting parameter k_m prevents their labels from growing too large.

The resulting labels are stored in a prefix tree (see Figure 3.5), a binary tree (also called 'trie') in which keys are not contained within nodes, but derived from the path that leads from the root to a node.

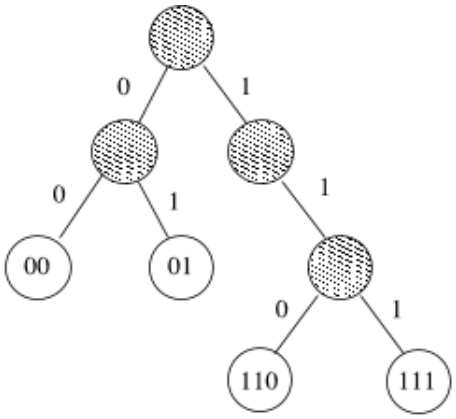

Figure 3.5: The figure shows an example of a possible prefix tree, constructed from four unique string labels "00","01","110","111". The data points are stored in the leafs, indicated with corresponding labels. Source: [Bawa et al., 2005].

Each level of the tree is associated with a different hashing function, sampled uniformly and with replacement from $\mathcal{H}$. Such a tree, an LSH-Tree, is the equivalent to an LSH-based hash-table and the composition of l trees is an LSH-Forest.

Given a query point p, finding close neighbors in a set of LSH-Trees T_1, T_2, ..., T_l is performed in two phases. First, in a top-down phase or descent, each tree T_i is searched for the leaf node with the best match to the binary string label of q. Labels are computed individually for each tree, starting from the root and extending the label until a leaf node is reached.

Inspecting the matches from all trees, the match with the longest prefix defines the tree-level x from which the bottom-up accumulation, the second phase, begins. While consecutively decreasing x until the root node is reached, matching leaf nodes across all trees are accumulated. Starting from the bottom-most level ensures that matches with longer prefixes are retrieved first. The process ends when M points have been gathered. These are searched for the closest match or m closest matches with the query point q using brute-force search and a distance function D.

3.3 Composition

In section 3.1 a model was introduced that can process textual data to produce high-dimensional vector representations. This section will describe how this model

was combined with the approximate nearest neighbor search, described in section 3.2, to retrieve textual responses, given the context of a conversation.

Encoding Step

The first step is to use a trained HRED model to encode dialogues D_n into embeddings, given some corpus $C = (D_1, D_2, ..., D_N)$ of size N. Each dialogue, consisting of M utterances $U_1, U_2, ..., U_M$, is saved to a database with the corresponding context embeddings $c_1, c_2, ..., c_M$ and utterance or response embeddings $h_1, h_2, ..., h_M$ with M being the number of turns of dialogue D_n. Formally, a dialogue is expressed as a set of M triples (U_m, h_m, c_m) (see Figure 3.6), where h_m is an encoded representation of utterance U_m and c_m is a summary of utterances $U_1, U_2, ..., U_m$, i.e, an embedding that represents the context of the conversation or multiple utterances up to turn-index m.

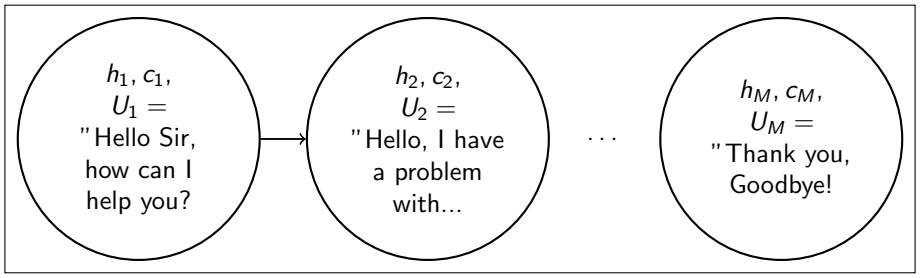

Figure 3.6: This figure shows a formal decomposition of a basic dialogue D_n with M turns. Each turn contains a textual utterance U_m, a response embedding h_m and a context embedding c_m.

Retrieval Step

Before being able to retrieve similar conversations or responses with a nearest neighbor algorithm, possible speed-up structures have to be initialized. The data the NNS algorithm operates on is the set of all context-embeddings or a sub-set of them if all do not fit into the main memory. Formally, an LSH-Forest indexing structure is initialized with a set of N tuples $(c_{n,m}, y_{n,m})$, with $c_{n,m}$ being a data point or context embedding and $y_{n,m}$ a label or pointer that links $c_{n,m}$ to the original triple (U_m, h_m, c_m) in dialogue D_n. As such, querying a context embedding will result in k labels pointing to the nearest neighbors in memory.

Going back to the original problem definition of this thesis, finding responses to a given context, suitable utterances can be found in similar conversations. If one increments a pointer $y_{n,m}$ returned by the approximate nearest neighbor model,

$y_{n,m+1}$ will point to the next conversational turn, i.e., to a possible response that was given in a similar conversation.

A real-case scenario is finding a response to a corpus-related textual conversation that might not be contained in the corpus. The first step in finding suitable responses is to encode the original conversation and use the last generated context embedding for querying. The second step is to increment the k resulting labels and retrieve the corresponding candidate responses $r_1, r_2, ..., r_k$. From the experiments, it became apparent that the closest neighbor does not necessarily contain the best response and as such, candidates are ranked according to a scoring function in the last step.

3.4 Candidate Selection

There are a couple of factors one can consider when scoring candidates: how much the retrieved context matches the original context, question-to-question similarity, answer relevance, and other text-based features. However, the scoring functions introduced in this section will solely be based on vector comparison metrics, such as the cosine similarity, as text-based comparison is less rewarding and more difficult and tedious to implement. For the sake of clarity, the query context embedding is defined as c_q, the textual candidates as $r_1, r_2, ..., r_k$, the context embeddings of candidates as $c_{r_1}, c_{r_2}, ..., c_{r_k}$, and the utterance embeddings of candidates as $h_{r_1}, h_{r_2}, ..., h_{r_k}$.

Context Relevance

The similarity of two conversations or the distance between a query context c_q and a candidate context c_{r_k} has, intuitively, a big impact on the retrieved answer, i.e, the more two questions are similar, the higher the probability that the answers are similar as well. If the cosine similarity has been chosen as the distance function D, the labels returned by the nearest neighbor search are already sorted by context-to-context distance. Formally, given a candidate response r_x and a query context c_q, the Context Relevance (CR) cost function is defined as:

$$cost_{CR}(r_x) = cos_{sim}(c_{r_x}, c_q). \tag{3.11}$$

Answer Relevance

By manual inspection of near neighbors, it became apparent that the correct answer is usually represented or almost captured in many topic-related candidates. Assuming that the most suitable topic for answering is dominantly represented amongst candidates, responses are ranked based on how much they capture the general topic. Formally, the cost of a response r_x is defined by the accumulated similarity between its respective embedding h_{r_x} and all other utterance embeddings:

$$cost_{AR}(r_x) = \frac{1}{k} \sum_{i=1}^{k} cos_{sim}(h_{r_x}, h_{r_i}) \tag{3.12}$$

Combining Context and Answer Relevance

The problem with the previous approach is that the candidates that are off-topic still contribute to the answer relevance cost. Therefore, in a pre-step, according to the previously described context relevance metric, the top n candidates are accumulated to represent the best general answer topic. In the next step, candidates are ranked based on their similarity to these n responses. Formally, combined Context and Answer Relevance (CAR) is defined as:

$$cost_{CAR}(r_x) = \frac{1}{n} \sum_{i=1}^{n} cos_{sim}(h_{r_x}, h_{r_i}), \tag{3.13}$$

with $n \leq k$.

An overview of the entire pipeline can be seen in Figure 1.1. In the following chapter, the used corpus will be introduced and the experiments that aim to compare the responses selected according to the aforementioned ranking methods and responses directly generated by the HRED model.

Chapter 4

Corpus and Experiments

First, a textual corpus of human-to-human customer service conversations will be introduced, considering pre-processing, structure and content of the data. Secondly, the experiments that evaluate the different candidate scoring metrics will be discussed.

4.1 Vodafone Online Customer Service Corpus

The Vodafone corpus was created during the making of this thesis, accessing archived conversations of the Dutch Vodafone online customer service. The entire creation process will be step-wise introduced in this section. As the corpus contains sensitive customer information, it will not be made publicly available, even though anonymization has been one of the pre-processes.

Chat Environment

Customers having problems with their phone, want to make contractual changes or experience other product related issues, often decide to talk with a Vodafone service agent through an online chat platform. Agents are not limited to providing general information but can, with correct authentication, perform administrative services, such as giving insights to active contracts and suggesting changes, locking and unlocking SIM cards, analyzing invoices, and other orders.

The customer information that the service agent has or the details an agent can see during a chat conversation are not contained within the corpus. This kind of hidden information is crucial to the understanding of service actions that the customer service can take or suggest. Without it, a model that operates solely on conversational text is not capable of performing all of those actions. This additional information, customer contracts, addresses, phone numbers and others, would need to be added as meta-tokens to the corpus or somehow provided to the model. However, such an annotation would be non-trivial to perform and fundamental changes

to the HRED design would, due to time restrictions, exceed the scope of this thesis. As such, the proposed model will have fundamental difficulties in answering a majority of questions entirely or precisely.

4.1.1 Conversation Retrieval

A Vodafone system automatically saves chat conversations to a database, archiving the last 13 months. The system is accessible through an online service interface, allowing one to download conversations manually for a certain time window. The number of retrieved conversations is limited by a constant value per query. Conversations exceeding the query limit would be dropped and, as such, a query was performed for each day individually, starting from the current day of the retrieval and going backwards for 13 months.

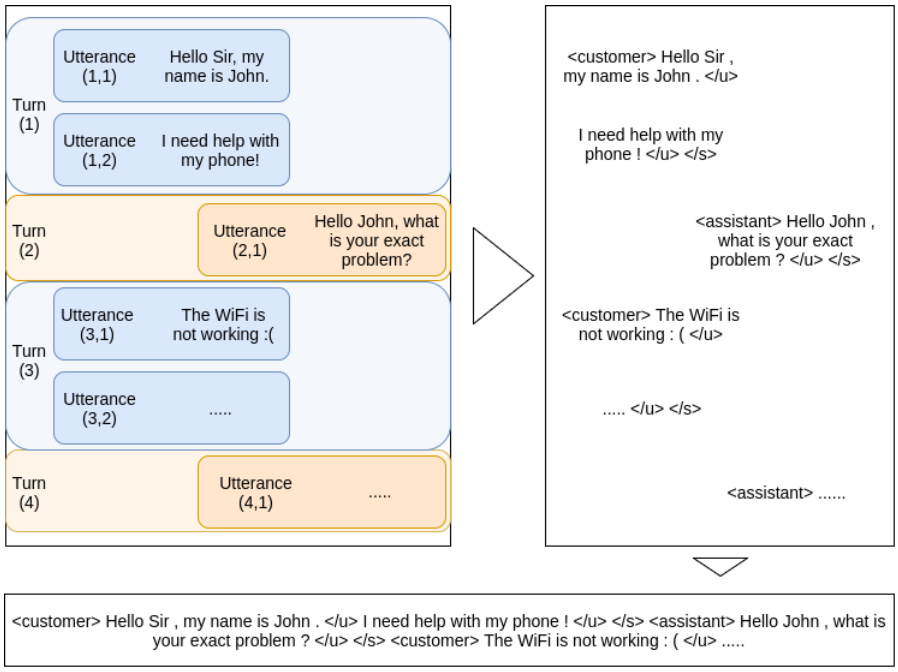

Figure 4.1: This figure shows how a common chat conversation is converted to a single string. The meta-tokens "<customer>" and "<assistant>" define who is currently typing, "</u>" indicates the end of an utterance and "</s>" is an end-of-turn token. The beginning and ending of a response, the text that is encoded into an utterance embedding, is defined by the end-of-turn token "</s>" and not the end-of-utterance token "</u>". Embeddings are still referred to as utterance embeddings to comply with the terminology of recent research.

The original conversations were provided in an XML format, which also contained chat-related meta information. The only information taken from those files were the actual conversations and an indication of who is currently speaking, either assistant or customer. Furthermore, the original decomposed representation of a conversation was converted into a single string, containing meta-tokens that indicate speaker turns, the end of an utterance and the end of a turn. An example conversion of a usual chat conversation is shown in Figure 4.1.

4.1.2 Pre-Processing

The HRED model requires a vocabulary containing each word that could possibly occur in the corpus. The vocabulary or dictionary is basically a hash table that maps word strings to an integer word-id. Furthermore, conversations used for training or encoding have to be converted into arrays of word-ids to comply with the HRED design. This, however, requires that a conversation can be unambiguously decomposed into its individual parts, i.e., each word, entity, or token has to be separated from others, e.g. by white space, and contained in the vocabulary. Pre-processing text in natural language processing (NLP) to comply with these requirements is referred to as tokenization, one of the most basic and crucial NLP tasks. The following two lines show a tokenization example:

```
"Hello, this is an example sentence! It won't be too long..."

"Hello" "," "this" "is" "an" "example" "sentence" "!" "It" "won't" "
    be" "too" "long" "..."
```

The initial number of retrieved dialogues is 472 thousand. However, there is a relatively large amount of conversations where one of the parties, either service agent or customer, did not respond at all, reducing the number of actual conversations to 443 thousand. Tokenization was performed on this set, using a NLP library, the python Natural Language Toolkit (NLTK) [Bird et al., 2009]. A list of properties of the corpus after performing tokenization can be seen in Table 4.1.

This tokenized corpus could have already been processed by the HRED model. However, it still contained many impurities, such as different languages amongst dialogues, arbitrary and possibly unique numbers and misspelled words that would blow up the size of the vocabulary and add additional noise to the training.

To detect different languages, the python library langdetect [Danilak, 2016], a port of Google's java language detection implementation [Shuyo, 2010] with a precison of more than 99%, has been used. The language detection of this library is based on an already trained Naive Bayes model that aims to find language specific features in text and to compute probabilities for the 55 supported languages. To find language specific features, the model utilizes precomputed probabilities of character n-grams belonging to a certain language.

Every conversation that was not clearly identified as Dutch text was filtered out of the corpus. Furthermore, to guarantee that the HRED model receives actual conversations for its training, conversations that have less than 5 turns have also been

Total # of dialogues	443,636
Total # of turns	6,961,247
Total # of utterances	11,109,988
Total # of words	130,068,804
Avg. # of words per dialogue	293.18
Avg. # of turns per dialogue	15.69
Avg. # of words per turn	18.68
Avg. # of utterances per dialogue	25.04
Avg. # of words per utterance	11.70

Table 4.1: This table shows the properties of the tokenized Vodafone corpus, containing conversations where both parties, service agent and customer, have at least responded once.

filtered out. Table 4.2 shows the number of dialogues for different combinations of filters.

At least two speakers	Dutch only	Min. 5 turns	Less than 3.0% unknown words	# of dialogues
✗	✗	✗	✗	472,517
✓	✗	✗	✗	443,636
✓	✓	✗	✗	423,504
✓	✗	✓	✗	401,964
✓	✓	✓	✗	385,816
✓	✓	✓	✓	384,897

Table 4.2: This table shows the effect of different filters on the final size of the corpus.

Data Cleaning, Filtering and Anonymization

The original corpus contains phone numbers, addresses, names, postal codes and other personal information. This data has been replaced for two reasons: First,

to remove a substantial amount of sensitive data and, secondly, the HRED model is not able to verify any of this information, due to previously described missing customer information, which a human service agent could access. Replacing an actual address or other concepts with a meta-token, e.g. "<street_name>" or "<city>", is considered to be beneficial for the performance of the HRED model. Word embeddings for actual street names or other rare character combinations will not be trained properly because there are not enough examples of them in the corpus. When using a meta-token such as "<street_name>" instead, many more training examples will contain its concept and the model can learn in which context a street name should appear. This is possible because the word embeddings of such concepts are also tuned during the training.

Before replacing words, each conversation was annotated using POS-tagging [van Miltenburg, 2016] to make spotting certain concepts and disambiguation easier. The reasoning behind this is that a concept such as names will be tagged as nouns and other words, tagged as a verb, adjective, etc., can be ignored and will not be falsely interpreted as a person's name. To replace sensitive information with meta-tokens that represent concepts, the following set of filter rules have been applied to the corpus:

- **Names**
  Using a database containing Dutch first and last names, any word that is contained in the database and tagged as a noun is replaced by either "<first_name>" or "<last_name>". Due to the high number of names in the database, last names could not be spotted reliably and their replacement would have resulted in too many false positives..

- **Addresses**
  Using another local database, containing dutch street names, city names, and postcodes, meta-tokens "<street_name>", "<city>", and "<post_code>" replace the respective words. City names and street names have to be spotted as nouns to be replaced.

- **Time**
  Strings that follow the hour and minute format "H:M" are replaced by "<time>".

- **Dates**
  All character sequences that can be interpreted as a date, e.g. "d/m/Y", "m/d/Y", and other valid formats, have been replaced by the "<date>" meta-token. Dates that could not be interpreted are replaced by other rules. Dates that are expressed by multiple tokens, e.g. "Jan 1 2017", are replaced by corresponding meta-tokens, e.g. "<month> <day> <year>".

- **Numbers**
  Any string that contains only digits and has not been filtered by another rule is replaced with "<number>". This will filter full or partial phone numbers, bank account numbers and other sensitive information. Most arbitrary numbers occur infrequently and as such, only add noise. The same holds for strings containing both, digits and text.

- **Text with numbers**
  If a word has not been filtered by another rule and contains both, numbers and characters, it is replaced by "<digits+text>". With the previous rules, the final corpus will not contain any numbers. This rule can also filter dates that have not been found previously.

Building the Dictionary

The dictionary was built by iterating over all conversation in the corpus, analyzing each word and adding words to the dictionary if they are not already contained. Furthermore, the occurrence of each word was counted, such that very infrequent words can be removed and replaced by an "<unk>" ('unknown') meta-token. This step has two advantages, the complexity of the model is reduced and a large amount of personal information, not found by previous steps, is removed automatically. The final size of the dictionary results from the minimal word occurrence, a threshold that defines how frequent a word needs to be in order to be kept. Figure 4.2 shows dictionary sizes for different thresholds and the influence the 'anonymization' or 'clean-up' step has on the final size. The previous step of anonymization reduced the initial dictionary size from 867947 to 411650.

In addition to the previously discussed dialogue filters, any conversation that has more than 3% unknown words, i.e., "<unk>" meta-tokens, has been removed. The intention was to remove cryptic dialogues that would add unnecessary noise to the training. However, there were only very few conversations, 919 specifically (see Table 4.2), that do not satisfy the 3% restriction.

The final corpus was generated by using a word occurrence threshold of 10, resulting in a dictionary size of 42892 and an average of 0.435% unknowns per dialogue. The entire statistics can be seen in Table 4.3.

Total # of dialogues	384,897
Total # of turns	6,571,902
Total # of utterances	10,461,677
Total # of words	122,325,433
Avg. # of words per dialogue	317.81
Avg. # of turns per dialogue	17.07
Avg. # of words per turn	18.65
Avg. # of utterances per dialogue	27.18
Avg. # of words per utterance	11.58

Table 4.3: This table shows the properties of the final Vodafone corpus.

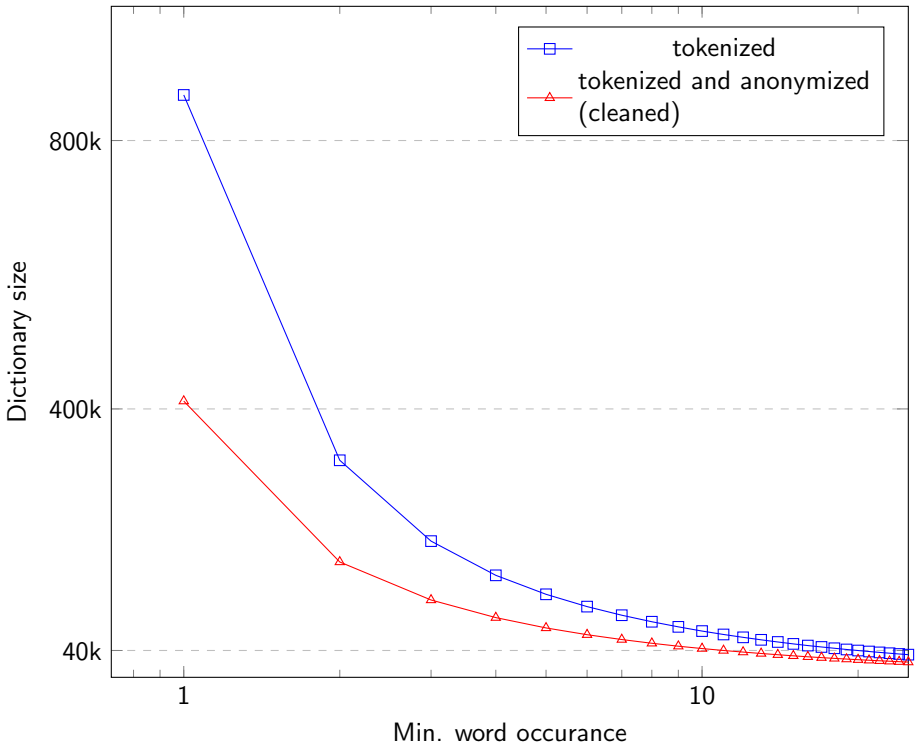

Figure 4.2: This figure compares the resulting dictionary sizes for different minimum word occurrences and for two differently processed versions of the Vodafone corpus. It is clearly visible that applying anonymization or a clean-up step reduces the initial and general dictionary size.

4.2 Experiments

Before showing the actual results, the different models and settings will be compared and a quantitative evaluation metric will be introduced.

4.2.1 Models

Going back to one of the research questions, how to combine global and specific domain knowledge embedded in word embeddings, three models have been compared, each initialized with a different set of word embeddings.

The first model, based only on local domain knowledge, received word embeddings trained with the gensim python library [Řehůřek and Sojka, 2010], a tool that, given a corpus, will train word embeddings specifically for that corpus.

The second model utilized word embeddings from [Tulkens et al., 2016], representing global domain knowledge. The embeddings were trained on a corpus consisting of 4 billion words, which was automatically generated by analyzing Dutch

websites.

The last model received word embeddings that are a combination of the two previously described sets. Both contain word embeddings with a feature-length of 320. However, the embeddings for the last model will have a length of 420, using a concatenation of global embeddings with 320 features and local embeddings with 100 features. An overview can be seen in Table 4.4.

Model name	Word embeddings	Feature length
$HRED_L$	Trained with gensim	320 (local knowledge)
$HRED_G$	Pretrained on large corpus	320 (global knowledge)
$HRED_{LG}$	Concatenation	420 (320 global + 100 local)

Table 4.4: This table shows the different word embeddings that have been used to initialize HRED models.

Furthermore, each of these models was used to encode the original corpus such that an approximate nearest neighbor model, based on LSH-Forests, could be generated to operate on embeddings. As the encoded corpus did not fit entirely into the main memory, a random subset with a size of 12% of the original corpus has been used to build the indexing structure of the LSH-Forest, i.e., the Nearest Neighbor Search operates only on 12% of all dialogues.

For each of the previously discussed models, three different candidate selection methods have been compared, Context Relevance (CR), Answer Relevance (AR) and Combined Context and Answer Relevance (CAR) (See Section 3.4). Also, performance measurements have been conducted for assistant and customer responses individually, as customer responses are more likely to be random or not context related.

In addition to the previously discussed retrieval-based models, the original generative approach has been included in the experiments for comparison. An overview of all models can be seen in Table 4.5.

4.2.2 Evaluation metric

A quantitative evaluation metric, the Recall@k measurement [Lowe et al., 2015], has been used to compare the ranking performance of models. Given a context, a set of n possible answers is presented to a model, which has to rank the answers by their likelihood of being the actual response. For a single evaluation sample, if the correct answer is ranked to be amongst the k best, the model succeeded. The overall performance of a model is defined as the ratio of correctly ranked answers to all answers, i.e., the percentage of correct answers that were ranked to be amongst the k best.

Model	Word embeddings Domain knowledge	Candidate selection
$HRED_L$-AR	local	Answer Relevance (AR)
$HRED_G$-AR	global	AR
$HRED_{LG}$-AR	local and global	AR
$HRED_L$-CR	local	Context Relevance (CR)
$HRED_G$-CR	global	CR
$HRED_{LG}$-CR	local and global	CR
$HRED_L$-CAR	local	AR & CR (CAR)
$HRED_G$-CAR	global	CAR
$HRED_{LG}$-CAR	local and global	CAR
$HRED_L$	local	Generative
$HRED_G$	global	Generative
$HRED_{LG}$	local and global	Generative

Table 4.5: A table that lists the different compared settings.

The evaluation samples were created from the test set, a part of the corpus that was 'held out', meaning that it has not been used for training or training validation. By iterating over the conversations in this set, an evaluation sample has been created for each individual turn or response, with the previous turns representing the context and the current turn or response being the ground truth. In addition to the actual response, a single example also contains $n - 1$ randomly sampled answers, which the model should preferably rank lower than the true answer. To make disambiguation harder, it was ensured that randomly sampled answers and the ground truth response, all start with the same speaker meta-token, `"<customer>"` or `"<assistant>"`.

Furthermore, to provide actual and standardized conversations, a dialogue was only used to generate evaluation samples if it contains at least 20 turns but not more than 40, resulting in a set of moderately long conversations. To increase the difficulty, the first and last 5 turns were removed. The reasoning behind this is that greetings and leave-takings usually occur at the beginning and ending of a conversation and that these are usually very generic and easy to respond to.

Considering the previously described criteria, 15000 evaluation samples (15000 individual turns) have been created by uniformly sampling dialogue turns from dialogues in the test set. Each instance contains a context of at least 5 turns, one true answer and 9 randomly sampled ones. A model had to retrieve or generate an answer based on the given context. The ranking has been created by computing the cosine similarity between the embedding of the suggested answer and the embeddings of the n possible responses. Each of the models in Table 4.5 was evaluated based on this set.

4.3 Results

The generative HRED approach and the different candidate selection methods, Context Relevance (CR), Answer Relevance (AR) and Context and Answer Relevance (CAR), have been compared for the three word embedding initialization approaches, $HRED_L$, $HRED_G$ and $HRED_{LG}$, individually. The performance of the $HRED_L$ model, focusing on specific domain knowledge, can be seen in the Figures 4.3 and 4.4 for assistant and customer responses respectively. Results for the $HRED_G$ model are shown in the Figures 4.5 and 4.6. The performance of the $HRED_{LG}$ model, combining local and global domain knowledge, can be seen in the Figures 4.7 and 4.8. A compact overview can be seen in Table 4.6.

	Predicting assistant responses			Predicting customer responses		
Model	R@1	R@2	R@5	R@1	R@2	R@5
$HRED_L$	$\mathbf{31.2 \pm 0.9}$	$\mathbf{45.8 \pm 0.8}$	$\mathbf{73.4 \pm 0.9}$	28.5 ± 0.7	$\mathbf{39.8 \pm 0.9}$	$\mathbf{66.1 \pm 0.8}$
$HRED_G$	29.9 ± 0.7	44.3 ± 0.9	71.1 ± 0.6	27.1 ± 0.7	37.7 ± 0.9	63.1 ± 0.9
$HRED_{LG}$	30.3 ± 1.0	44.2 ± 0.9	70.6 ± 0.7	$\mathbf{28.9 \pm 1.1}$	$\mathbf{39.8 \pm 1.1}$	64.6 ± 1.2
$HRED_L$-CR	33.4 ± 0.7	48.0 ± 0.8	75.4 ± 0.6	32.8 ± 1.0	47.5 ± 1.0	73.5 ± 0.9
$HRED_G$-CR	34.6 ± 1.1	50.0 ± 1.0	76.2 ± 0.7	32.9 ± 0.8	47.3 ± 0.7	74.2 ± 0.8
$HRED_{LG}$-CR	33.9 ± 0.9	48.8 ± 0.8	74.8 ± 0.8	32.5 ± 0.9	48.1 ± 0.7	74.8 ± 0.6
$HRED_L$-AR	39.6 ± 0.7	55.7 ± 0.9	81.1 ± 0.7	40.0 ± 1.1	55.8 ± 1.0	80.4 ± 0.8
$HRED_G$-AR	42.7 ± 0.9	58.5 ± 0.8	82.5 ± 0.7	$\mathbf{41.0 \pm 0.9}$	$\mathbf{56.9 \pm 0.7}$	$\mathbf{81.5 \pm 0.6}$
$HRED_{LG}$-AR	43.0 ± 0.8	59.4 ± 0.8	$\mathbf{82.7} \pm 0.8$	40.1 ± 0.9	55.7 ± 0.9	80.0 ± 0.9
$HRED_L$-CAR	41.3 ± 0.8	57.2 ± 0.8	81.2 ± 0.5	39.9 ± 1.0	55.3 ± 1.0	79.6 ± 0.6
$HRED_G$-CAR	$\mathbf{44.0 \pm 0.7}$	$\mathbf{59.8 \pm 0.9}$	82.6 ± 0.7	40.9 ± 0.7	56.8 ± 0.7	80.6 ± 0.7
$HRED_{LG}$-CAR	43.8 ± 0.7	59.5 ± 0.8	82.6 ± 0.7	39.4 ± 0.7	55.1 ± 0.9	79.6 ± 0.6

Table 4.6: Overall comparison of model precisions (in %). Confidence intervals ($\pm$95%) are shown next to the average performance.

As expected, for the majority of settings, models can predict assistant responses easier than customer responses. The CAR candidate selection method outperforms all other techniques when predicting assistant responses. However, the performance of customer response prediction is slightly dominated by AR. A reason for this could be that, as discussed earlier, customers often reply with new questions that might not be context related, making answer relevance more important than context relevance.

Furthermore, it can be seen that initializing the HRED model with word embeddings containing global domain knowledge results in the best performance for candidate selection approaches. However, combining global and local domain knowledge has not led to the desired improvements. This can be explained as follows: The computational graph of the HRED model that defines its training also includes tuning the word embeddings. As such, during the training, the word embeddings are already altered to encode local domain knowledge, even if they were only initialized with embeddings containing global domain knowledge. Adding additional feature-length will in the worst case only add complexity to the model.

The performance of the generative approaches, $HRED_L$, $HRED_G$, and $HRED_{LG}$, are relatively similar. However, $HRED_L$ slightly outperforms the others. This is contradictory, considering that the candidate selection methods, CR, AR and CAR, clearly perform better on embeddings generated by $HRED_G$ and $HRED_{LG}$ (See Table 4.6). A reason for this could be that utilizing global domain knowledge to generate an answer is more difficult than using specific domain knowledge. Especially for a very homogeneous corpus, giving standard answers can work. Nonetheless, similarity comparisons, used by the NNS approaches, could still benefit from richer embeddings.

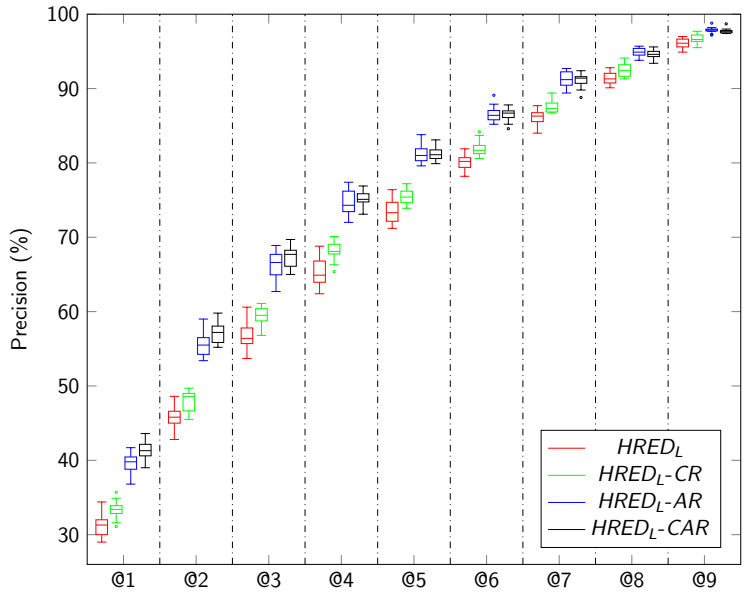

Figure 4.3: Recall@k performance of the $HRED_L$ model in predicting assistant responses.

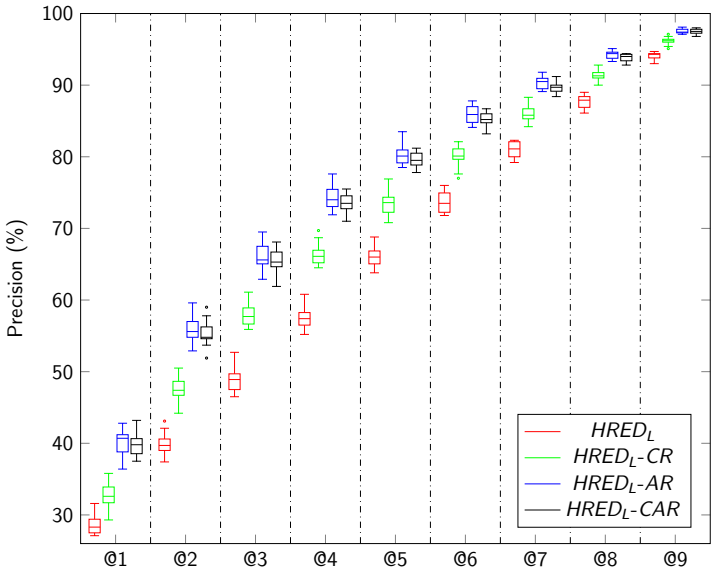

Figure 4.4: Recall@k performance of the $HRED_L$ model in predicting customer responses.

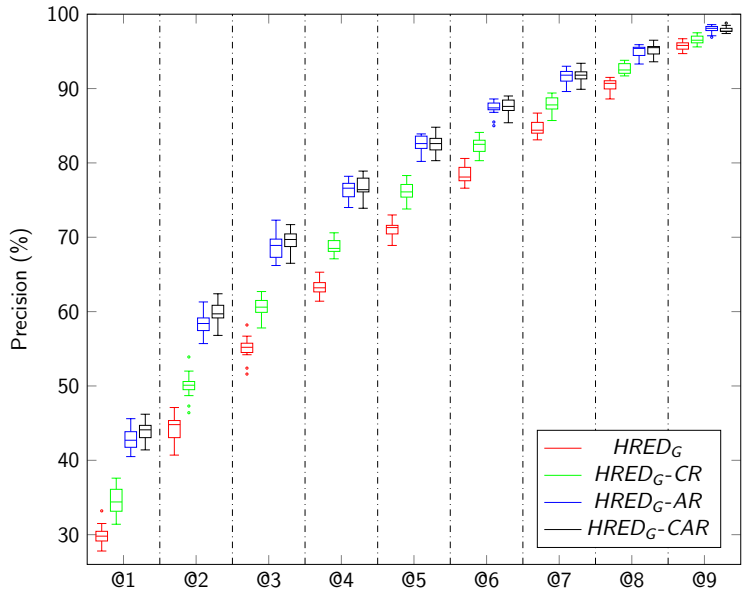

Figure 4.5: Recall@k performance of the $HRED_G$ model in predicting assistant responses.

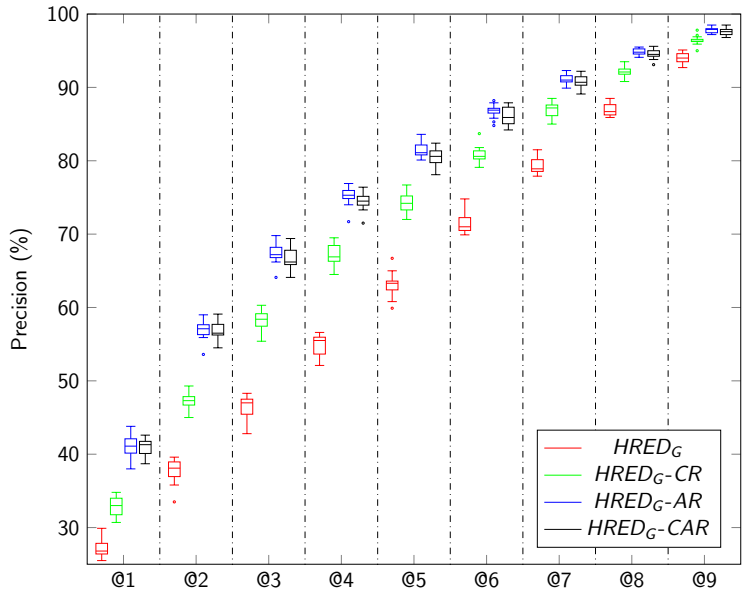

Figure 4.6: Recall@k performance of the $HRED_G$ model in predicting customer responses.

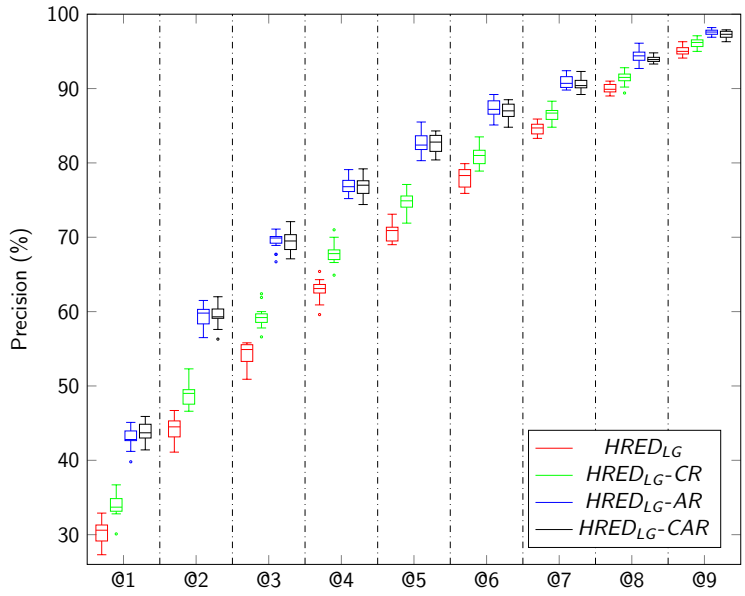

Figure 4.7: Recall@k performance of the *HRED$_{LG}$* model in predicting assistant responses.

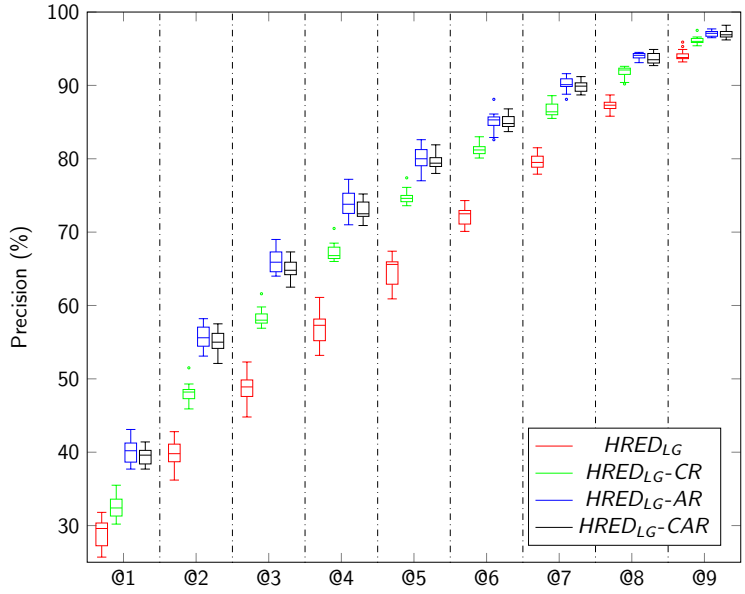

Figure 4.8: Recall@k performance of the *HRED$_{LG}$* model in predicting customer responses.

Chat Examples

Two examples of how the retrieval based approach would respond to a given context can be seen in Tables 4.7 and 4.8. For comparison, the actual answer and the answer generated by the $HRED_G$ model are shown in addition to the three best candidates of the $HRED_G$-CAR model. The original Dutch text was translated into English, capturing the actual meaning with as least alteration as possible. Tables 4.9 and 4.10 show the unmodified examples.

The first example in Table 4.7 shows a conversation about a customer who has a damaged phone and it is not entirely clear if the customer caused the damage or if the phone had an internal issue. This information has consequences on whether repair-under-warranty applies. In the actual response, the agent asks if the customer has an insurance. The first two responses proposed by the retrieval based approach ask the same question. However, there are some differences. The first introduces additional insurance related information to the customer. The second candidate response suggests sending the phone somewhere for repairs and shows an issue that is common to the retrieval based approach, namely that the sentence structure does not fit the previous conversation, but the answer is context related. The answer proposed by the $HRED_G$ model is generic but valid. However, through manual inspection, it became apparent that the $HRED_G$ model often replies with the same generic responses, an issue that suits the findings of [Serban et al., 2016b, Serban et al., 2016c] and has been improved by more recent research, e.g. the VHRED model.

The second example in Table 4.8 is about a customer that recently extended his subscription and received a bill that is too high, a very basic and common chat service conversation. In the actual response, the service agent tells the customer that he will look into the issue and proposes a phone number for the phone in question. In this case, the response generated by the $HRED_G$ model, being almost the same as the actual one, suits the conversation perfectly, even if it is generic.

The first two retrieved candidates do fit the conversation but they are either poorly chosen or suffer from the previously described issue regarding sentence structure. Both contain asking the customer if he altered his subscription, which he already stated in the last turn. However, these responses could also be seen as asking for more information, which could be achieved by a simple change in the sentence structure. The third candidate is a good response, asking the customer if this is the first bill after the extension of his subscription.

Context
<customer> my phone is damaged, it is new, do i get a new phone? </u> <assistant> did the phone fall? </u> <customer> no </u> <assistant> when did you got the phone? </u> <customer> I had the phone in my pocket, just when taking it out I saw that maybe it was scratched by my keys </u> <month> </u>
Actual Response
<assistant> have you got an insurance? </u>
HRED$_G$
<assistant> I will gladly check it for you </u> does it concern <number> ? </u>
HRED$_G$-CAR Candidates (best three)
<assistant> you have <number> years warranty for your device. internal device issues are covered. if you haven't damaged the phone yourself, it falls under the regular warranty. </u> do you have an insurance with us? then you are insured either way. </u> <assistant> here they actually can check your device and if necessary, send it in for repair. </u> oh ... you have an insurance? </u> <assistant> You can then send your device in for repair. </u> if you have the device replacement service in your subscription your device will be <unk> and you will receive a replacement device. In case you don't have it? </u> then you can bring your device </u> to a Vodafone shop. </u>

Table 4.7: An example showing the three best responses retrieved by the *HRED$_G$-CAR* model to a given context. For comparison reasons, the answer generated by the *HRED$_G$* model and the actual response are shown as well. The text was translated into English with as least alteration as possible. The dutch original can be seen in Table 4.9.

Context
<customer> hello, why is my invoice this month <decimal> </u> <assistant> hi <first_name>! </u> I can check that for you! </u> <customer> great! </u> <assistant> is it higher than expected? </u> <customer> yes it is around <number> I extended my subscription last month, and I think there was an error </u>
Actual Response
<assistant> which is never nice indeed :(</u> ahh ok! </u> I will check your information! does it concern this number: <number> ? </u>
$HRED_G$
<assistant> I will look for you into the matter. </u> does it concern <digits+text>? </u>
$HRED_G$-CAR Candidates (best three)
<assistant> right. have you just changed your subscription? </u> <assistant> have you recently extended your subscription? </u> <assistant> is this your first invoice after the extension? </u>

Table 4.8: Another example of how retrieved answers look like to a given context. In this example, the generic answer generated by the $HRED_G$ model suits the conversation. The original Dutch text can be seen in Table 4.10.

Context
<customer> me telefoon is beschadigd, deze is net nieuw krijg ik een nieuw toestel? </u> <assistant> is het toestel gevallen? </u> <customer> nee </u> <assistant> wanneer heb je toestel ontvangen ? </u> <customer> ik had de telefoon in me tas gewoon bij het eruit halen zag ik dat misschien tegen me sleutels aangekomen </u> <month> </u>
Actual Response
<assistant> heb je een verzekering? </u>
HRED$_{G}$
<assistant> ik kijk het graag voor je na. </u> gaat het om <number> ? </u>
HRED$_{G}$-CAR Candidates (best three)
<assistant> je hebt <number> jaar toestelgarantie. hier vallen interne toestelproblemen onder. als je zelf geen schade aan het toestel hebt gemaakt, valt dit onder de normale toestelgarantie. </u> heb je een verzekering bij ons afgesloten? ben je sowieso verzekerd. </u> <assistant> hier kunnen ze namelijk je toestel nakijken en mocht het nodig izjn opsturen voor reparatie. </u> oh ... heb je een garant verzekering? </u> <assistant> je kunt je toestel dan opsturen ter reparatie. </u> als je een vervangende toestel service bij je abonnement hebt wordt je toestel <unk> en ontvangt je een vervangend toestel </u> heb je dit niet? dan kun je jouw toestel </u> inleveren bij een vodafone winkel. </u>

Table 4.9: The Dutch original of the example shown in table 4.7.

Context
<customer> hallo, waarom is mijn rekening deze maand <decimal> </u> <assistant> hoi <first_name>! </u> dat kan ik voor je nakijken! </u> <customer> top! </u> <assistant> is hij hoger dan verwacht? </u> <customer> ja hij hoort rond de <number> te zijn. ik heb vorige maand mijn abbonement verlengt, en ik denk dat daar iets fout is gegaan </u>
Actual Response
<assistant> dat is nooit leuk inderdaad :(</u> ahh ok! </u> ik neem je gegevens erbij! gaat het om dit nummer: <number> ? </u>
HRED$_{G}$
<assistant> ik ga voor je kijken wat er aan de hand is. </u> gaat het om <digits+text>? </u>
HRED$_{G}$-CAR Candidates (best three)
<assistant> klopt. heb je pas je abonnement veranderd? </u> <assistant> heb je onlangs je abonnement verlengd? </u> <assistant> is dit de eerste factuur na een verlenging? </u>

Table 4.10: The Dutch original of the example shown in table 4.8.

Chapter 5

Conclusion and Future Research

End-to-End Dialogue Systems are relatively new and most architectures are far away from being ready for deployment. Considering the difficulty of designing a system that can intelligently interact with humans, implementing truly robust and versatile Dialogue Systems will require more years of research.

The architecture proposed in this thesis can be seen as a combination of end-to-end and modular Dialogue System. The used retrieval based approach, utilizing dialogue and utterance embeddings which were trained end-to-end, has been shown to outperform the generative approach of the HRED model, even though both approaches operate on the same embeddings and only 12% was used for the NNS. However, retrieved answers, possibly being more context related, often do not fit the previous conversation in terms of their sentence structure.

Another core issue was discussed earlier: To truly understand a service conversation, also for humans, it would be necessary to have the additional context information, the data the service agent has access to, included in the corpus or presented to the model in a different way.

More recently proposed end-to-end systems, the VHRED model and Multiresolution Recurrent Neural Networks [Serban et al., 2016a], both being an improved version of the HRED model, are raising another question: Will one of these models outperform the proposed retrieval based approach, even though all operate on the same embeddings, i.e., at which point does the generative approach benefit from the embeddings quality more than the retrieval-based?

The proposed model is not capable of conducting the same tasks as a human service agent, however, it can be used as a support system, suggesting a ranked set of possible responses to an agent. Through a reward system, analyzing the feedback of agents, e.g. their acceptance or rejection of responses, the proposed approach

can be improved iteratively.

Another possibly valuable side product of this research addresses the following issue: Script-based systems often use strict rules and keyword spotting to transition between states. Instead of defining rules, which can be tedious, one could define a sentence or the "meaning", i.e., an utterance embedding, that should lead to a transition. As such, a textual user input, e.g. `"my invoice is too high"`, has to be encoded into an utterance embedding and compared to utterance and transition pairs, e.g. `("I have a huge bill", A->B)` or `("I have an issue with my phone", A->C)`, previously defined in a script. By measuring the cosine similarity between embeddings, the most suitable transition is chosen, e.g. `("I have a huge bill", A->B)`. Hence, to define a transition, one does not have to specify any possible word combination anymore but can just define the "meaning" that is looked for.

Going back to the last research question, *How can specific and general domain knowledge be encoded into an embedding and how does this affect the performance of models that build on them?*, the experiments have revealed that concatenating word embeddings with global and specific domain knowledge did not improve performance. The reason for this is expected to be found in the training of the HRED model. Even if initialized with word embeddings containing only global domain knowledge, word embeddings will be tuned to encode specific domain knowledge during the training.

The second research question, *As the NNS returns multiple nearest neighbors, how can those be scored such that the best response can be found?*, has been addressed with the proposed candidate selection methods, CR, AR and CAR. The results show that these outperform the generative approach.

The first research question, *How well do the dialogues returned by NNS fit to the context of the conversation at hand and how is this affected by the quality of embeddings?*, has been indirectly and directly addressed by the quantitative and qualitative measurements of the experiments. However, as only human evaluation is considered to be a robust measurement for dialogue systems, the experiments only give an abstract overview of the system's performance. Generally, the semantic distance between retrieved conversations and the conversation at hand is decent. There is also a difference of how the two approaches, retrieval based and generative, respond to the quality of embeddings. Including global domain knowledge in word embeddings did not improve the performance of the generative approach but candidate selection methods performed significantly better.

Future work

Dialogue and utterance embeddings have been only generated by one source, the HRED model. For comparison reasons, it would be interesting to see the perfor-

mance of other encoding approaches, such as averaging over word embeddings. Additionally, the embeddings generated by the recently proposed VHRED model and Multiresolution Recurrent Neural Networks are of a higher quality and likely to improve the performance of the proposed approach.

An interesting research topic is to actually simulate conversations with tree search. The idea is to use context embeddings as state or tree node representations and utterance embeddings as actions or connections leading from one state to another. The concept is inspired by how board games are approached with heuristic tree search algorithms, aiming to find desirable states through an exploration/exploitation strategy. Using responses retrieved by NNS as a set of actions, the search tree can explore possible paths and score responses based on the quality of simulated conversations.

Another possible research topic regards the previously discussed issue of the corpus, the additional context information a service agent can see and that is not contained in the conversation. Considering the size of corpora used to train end-to-end systems (usually around $500,000$ conversations), manual annotation can be very slow and costly. Finding an automated approach to make addresses, contractual details and other features accessible to an end-to-end Dialogue System is an interesting and rewarding task. Furthermore, to make such a system more useful, it would also be necessary for the model to be able to perform actions and database requests. This, however, would also require an automated annotation procedure that induces actions from conversations or from additional meta-information.

Solving the aforementioned problems will facilitate the deployment of end-to-end Dialogue Systems in online chat service environments, improving robustness, utility and customer experience.

Bibliography

[Banchs and Li, 2012] Banchs, R. E. and Li, H. (2012). IRIS: a chat-oriented dialogue system based on the vector space model. In *Proceedings of the ACL 2012 System Demonstrations*, pages 37–42. Association for Computational Linguistics.

[Bates, 1995] Bates, M. (1995). Models of natural language understanding. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 92, pages 9977–9982. National Academy Press.

[Bawa et al., 2005] Bawa, M., Condie, T., and Ganesan, P. (2005). LSH Forest: self-tuning indexes for similarity search. In *Proceedings of the 14th international conference on World Wide Web*, pages 651–660. ACM.

[Bengio, 2012] Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer.

[Bengio et al., 2013] Bengio, Y., Boulanger-Lewandowski, N., and Pascanu, R. (2013). Advances in optimizing recurrent networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8624–8628. IEEE.

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. In *IEEE transactions on neural networks*, volume 5, pages 157–166. IEEE.

[Bird et al., 2009] Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit*. "O'Reilly Media, Inc.".

[Bottou, 2012] Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer.

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the Empiricial Methods in Natural Language Processing*, pages 1724–1734. Association for Computational Linguistics.

[Chung et al., 2014] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Deep Learning Workshop*.

[Collobert and Weston, 2008] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

[Danilak, 2016] Danilak, M. (2016). *Port of Google's language-detection library to Python*. Retrieved March 23, 2017 from the World Wide Web: `https://github.com/Mimino666/langdetect`.

[Dowding et al., 1993] Dowding, J., Gawron, J. M., Appelt, D., Bear, J., Cherny, L., Moore, R., and Moran, D. (1993). Gemini: A natural language system for spoken-language understanding. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 54–61. Association for Computational Linguistics.

[Gers and Schmidhuber, 2000] Gers, F. A. and Schmidhuber, J. (2000). Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194. IEEE.

[Har-Peled et al., 2012] Har-Peled, S., Indyk, P., and Motwani, R. (2012). Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. In *Theory of computing*, volume 8, pages 321–350.

[Hinton et al., 2012] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. In *IEEE Signal Processing Magazine*, volume 29, pages 82–97.

[Hochreiter et al., 1997] Hochreiter, S., urgen Schmidhuber, J., and Elvezia, C. (1997). LONG SHORT-TERM MEMORY. In *Neural Computation*, volume 9, pages 1735–1780.

[Jozefowicz et al., 2015] Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An Empirical Exploration of Recurrent Network Architectures. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 2342–2350.

[Jurafsky and James, 2000] Jurafsky, D. and James, H. (2000). *Speech and language processing an introduction to natural language processing, computational linguistics, and speech*. Pearson Education.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. E. (2015). Deep learning. In *Nature: International weekly journal of science*, volume 521, pages 436–444. Macmillan.

[Lopez et al., 2008] Lopez, R., Balsa-Canto, E., and Oñate, E. (2008). Neural networks for variational problems in engineering. In *International Journal for Numerical Methods in Engineering*, volume 75, pages 1341–1360. Wiley Online Library.

[Lowe et al., 2015] Lowe, R., Pow, N., Serban, I. V., and Pineau, J. (2015). The Ubuntu Dialogue Corpus: A Large Dataset for Research in Unstructured Multi-Turn Dialogue Systems. In *16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*.

[Manning and Schütze, 1999] Manning, C. D. and Schütze, H. (1999). Foundations of statistical natural language processing. *The MIT Press*.

[Markoff and Mozur, 2015] Markoff, J. and Mozur, P. (2015). For sympathetic ear, more chinese turn to smartphone program. *NY Times*.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*.

[Mikolov et al., 2010] Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent Neural Network Based Language Model. In *Eleventh Annual Conference of the International Speech Communication Association*.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[Olah, 2015] Olah, C. (2015). Understanding lstm networks [Blog post]. Retrieved March 10, 2017 from the World Wide Web: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. In *Journal of Machine Learning Research*, volume 12, pages 2825–2830.

[Pietquin and Hastie, 2013] Pietquin, O. and Hastie, H. (2013). A survey on metrics for the evaluation of user simulations. In *The knowledge engineering review*, volume 28, pages 59–73. Cambridge Univ Press.

[Řehůřek and Sojka, 2010] Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. `http://is.muni.cz/publication/884893/en`.

[Schmidhuber, 2015] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. In *Neural networks*, volume 61, pages 85–117. Elsevier.

[Serban et al., 2016a] Serban, I. V., Klinger, T., Tesauro, G., Talamadupula, K., Zhou, B., Bengio, Y., and Courville, A. (2016a). Multiresolution Recurrent Neural Networks: An Application to Dialogue Response Generation. *arXiv preprint arXiv:1606.00776*.

[Serban et al., 2015] Serban, I. V., Lowe, R., Charlin, L., and Pineau, J. (2015). A survey of available corpora for building data-driven dialogue systems. *arXiv preprint arXiv:1512.05742*.

[Serban et al., 2016b] Serban, I. V., Sordoni, A., Bengio, Y., Courville, A., and Pineau, J. (2016b). Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 3776–3783. AAAI Press.

[Serban et al., 2016c] Serban, I. V., Sordoni, A., Lowe, R., Charlin, L., Pineau, J., Courville, A., and Bengio, Y. (2016c). A Hierarchical Latent Variable Encoder-Decoder Model for Generating Dialogues. *arXiv preprint arXiv:1605.06069v3*.

[Shah et al., 2016] Shah, P., Hakkani-Tür, D., and Heck, L. (2016). Interactive reinforcement learning for task-oriented dialogue management. In *NIPS 2016 Deep Learning for Action and Interaction Workshop*.

[Shawar and Atwell, 2007] Shawar, A. and Atwell, E. (2007). Chatbots: are they really useful? In *Journal for Language Technology and Computational Linguistics*, volume 22, pages 29–49. GSCL German Society for Computational Linguistics.

[Shuyo, 2010] Shuyo, N. (2010). Language Detection Library for Java. Retrieved March 27, 2017 from the World Wide Web: `http://code.google.com/p/language-detection/`.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of Go with deep neural networks and tree search. In *Nature*, volume 529, pages 484–489. Nature Publishing Group.

[Slaney and Casey, 2008] Slaney, M. and Casey, M. (2008). Locality-sensitive hashing for finding nearest neighbors [lecture notes]. In *IEEE Signal Processing Magazine*, volume 25, pages 128–131.

[Sordoni et al., 2015] Sordoni, A., Bengio, Y., Vahabi, H., Lioma, C., Grue Simonsen, J., and Nie, J.-Y. (2015). A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 553–562. ACM.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[Tulkens et al., 2016] Tulkens, S., Emmery, C., and Daelemans, W. (2016). Evaluating Unsupervised Dutch Word Embeddings as a Linguistic Resource. In Chair), N. C. C., Choukri, K., Declerck, T., Grobelnik, M., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France. European Language Resources Association (ELRA).

[van Miltenburg, 2016] van Miltenburg, E. (2016). *Simple perceptron tagger trained using the NLTK on the NLCOW14 corpus*. Retrieved March 24, 2017 from the World Wide Web: `https://github.com/evanmiltenburg/Dutch-tagger`.

[Vinyals and Le, 2015] Vinyals, O. and Le, Q. (2015). A neural conversational model. In *International Conference on Machine Learning: Deep Learning Workshop*.

[Weber et al., 1998] Weber, R., Schek, H.-J., and Blott, S. (1998). A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Very Large Data Bases (VLDB) Conference*, volume 98, pages 194–205.

[Weizenbaum, 1966] Weizenbaum, J. (1966). ELIZAa computer program for the study of natural language communication between man and machine. volume 9, pages 36–45. ACM.

[Wen et al., 2015] Wen, T.-H., Gašic, M., Kim, D., Mrkšic, N., Su, P.-H., Vandyke, D., and Young, S. (2015). Stochastic Language Generation in Dialogue using Recurrent Neural Networks with Convolutional Sentence Reranking. In *16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, page 275.

[Williams and Zipser, 1989] Williams, R. J. and Zipser, D. (1989). A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. In *Neural Computation*, volume 1, pages 270–280.

[Young et al., 2013] Young, S., Gašić, M., Thomson, B., and Williams, J. D. (2013). POMDP-based Statistical Spoken Dialogue Systems: a Review. In *Proceedings of the IEEE*, volume 101, pages 1160–1179.