Deep Learning for User Simulation in a Dialogue System

Florian Lennard Kreyssig

Supervisor: Dr. Milica Gašić

Department of Engineering
University of Cambridge

A Fourth-year Project Report
Submitted in Partial Fulfilment of the Requirements
For the Degree of
Master of Engineering
In
Information and Computer Engineering

Emmanuel College

May, 2018

Technical Abstract: Deep Learning for User Simulation in a Dialogue System

Florian L. Kreyssig, Emmanuel College

Task-Oriented Spoken Dialogue Systems (SDS) help users achieve goals such as finding restaurants or booking flights. Teaching a system how to respond appropriately in a task-oriented setting is non-trivial. In state of the art systems, this dialogue management task is often formulated as a reinforcement learning problem. In this framework, the system learns by a trial and error process governed by a reward function. A typical approach to defining the reward function for a task-oriented SDS is to apply a small per-turn penalty to encourage short dialogues and to give a large positive reward at the end of each successful interaction. Using reinforcement learning the SDS is trained to act such that is maximises the cumulative future reward over the course of a dialogue. The majority of reinforcement learning algorithms used for the training of dialogue strategies require interaction with an environment, in this case a user. This precludes the use of most corpus based machine learning algorithms. A solution is to use a corpus of human-computer dialogue to train a user simulator, which then acts as a synthetic dialogue partner for the learning system. Therefore, no interaction with real users is needed and an unlimited number of dialogues can be created with each dialogue typically being faster than a dialogue with a human. This also allows dialogue strategies that are not in the recorded corpus to be explored.

Previously proposed user simulators have several drawbacks. The majority model user behaviour on the semantic level rather than the word level. First, this prevents dialogue systems using a joint module for natural language understanding and belief tracking to be trained in a simulated environment. Second, trainable user simulators need semantic annotations for every user utterance. Furthermore, many user simulators lack an explicit goal representation, which means that at the end of a dialogue it is impossible to compare the user's goal with the proposal of the SDS in order to calculate the reward required for reinforcement learning. Other simulators insufficiently modelled the history, required too much engineering effort or domain specific knowledge, and many were based on a set of rules with parameters that were handpicked rather than learned. Much of human user behaviour is ill-understood and therefore there is a limitation to the performance of user simulators with handpicked parameters. The previous de facto state of the art in user simulation was the Agenda-Based User Simulator (ABUS). The ABUS is based on a set of rules and a set of highly interpretable parameters, and its output is in a semantic form.

The principal contribution contained in this work is a novel user simulator, termed Neural User Simulator (NUS). The NUS generates natural language and it is explicitly conditioned on a goal, which can change during the course of a dialogue. The core of the NUS is a neural sequence-to-sequence model, which allows it to efficiently model the dialogue history by learning what constitutes a good user state. The input to the neural sequence-to-sequence model are hand designed embeddings of the SDS's output. Furthermore, its entire behaviour is learned from a corpus of human-computer dialogues and only a modest amount of work is needed to define how the embeddings are created, reducing the necessary engineering effort to a minimum.

In comparison to much of the past work on user simulation, the NUS is used to train the policy of a reinforcement learning based Spoken Dialogue System. This was often omitted and instead proposed user simulators were evaluated by computing a statistical measure of similarity between the outputs of the user simulator and a real user on a test set. However, new user simulators are introduced primarily with the motivation that training with the proposed user simulator yields an improved policy. Therefore, this project compares the NUS to the ABUS by evaluating the policies that were trained using the simulators. Cross-model evaluation is performed i.e. training on one simulator and testing on the other. Furthermore, the trained policies are tested on real users. In both evaluation tasks the NUS outperformed the ABUS.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this report are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This report is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and acknowledgements. This report contains fewer than 12,000 words including figures and appendices, but not counting the title page, bibliography and technical abstract. Some of the material included in this report has been accepted for publication at the Annual Meeting of the Special Interest Group on Discourse and Dialogue (Kreyssig et al., 2018a).

Florian Lennard Kreyssig
May, 2018

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Milica Gašić, for providing me with the opportunity to work on this project. By endowing me with the freedom to pursue my own ideas, but also keeping me organized and focused she made this work possible.

I am also in debt of Prof. Phil Woodland, who initially introduced me to Deep Learning by accepting me as an intern in his group for the past two summers. I am also grateful to Dr. Chao Zhang for his continuous support during these internships, the experience of which was invaluable during the course of this project.

I would also like to thank Dr. Iñigo Casanueva for his help with the PyDial Toolkit, Paweł Budzianowski for helping me set up the human evaluation and the rest of the Dialogue Systems Group for making the hours of work in the office such an enjoyable experience.

Throughout my undergraduate studies, I was supported by the Studienstiftung des Deutschen Volkes, which I am grateful for.

To Gemma, thank you for your company, patience and support.

Finally, the biggest thanks go to my family for all their love and encouragement. Without them, I would not be here.

Contents

1 Introduction **6**

1.1 Overview 6

1.1.1 Task-Oriented Dialogue Systems 6

1.1.2 Simulated training of a Dialogue System 8

1.2 Motivation 8

1.3 Contributions 10

2 Statistical User Modelling for Simulation-Based Dialogue Strategy Learning **11**

2.1 Overview 11

2.2 Intention Level User Modelling 12

2.3 Word Level User-Modelling 14

3 Deep Learning **16**

3.1 Feedforward Neural Networks 16

3.2 Parameter Optimization 18

3.2.1 Cross-Entropy Loss 18

3.2.2 Back-propagation 19

3.3 Recurrent Neural Networks 20

3.3.1 Long Short-Term Memory Networks 20

3.3.2 Natural Language Generation using SC-LSTM 21

4 Neural User Simulator **24**

4.1 Goal Generator 24

4.1.1 Goal-Labels in DSTC2 26

4.2 Feature Extractor 26

4.3 Sequence-To-Sequence Model 27

4.4 Training 29

5 Experiments **30**

5.1 DSTC2 Dataset 30

5.2 Experimental Setup . . . 30
5.2.1 Training of Dialogue Policies . . . 31
5.2.2 Testing with simulated users . . . 31
5.2.3 Testing with real users . . . 32
5.3 Experimental Results . . . 32
5.3.1 Cross-Model Evaluation . . . 32
5.3.2 Human Evaluation . . . 34

6 Conclusions 35
6.1 Summary . . . 35
6.2 Limitations . . . 36
6.3 Future Work . . . 37

Bibliography 38

Chapter 1

Introduction

1.1 Overview

Spoken Dialogue Systems (SDS) allow human-computer interaction using natural speech. The use of Spoken Dialogue Systems in commercial applications has become common with products such as Amazon's Echo, Google's Home, and Apple's Siri and HomePod. Whilst these systems can sometimes hold simple conversations or tell jokes, the ability of such systems to achieve specific goals is rather limited. Therefore this work focuses on *task-oriented* dialogue; helping users achieve specific goals such as booking restaurants or booking flights. Systems for task-oriented dialogue are often trained by interacting with a synthetic dialogue partner: a *user simulator*. Improving the state of the art in user simulation was the ambition of this project.

1.1.1 Task-Oriented Dialogue Systems

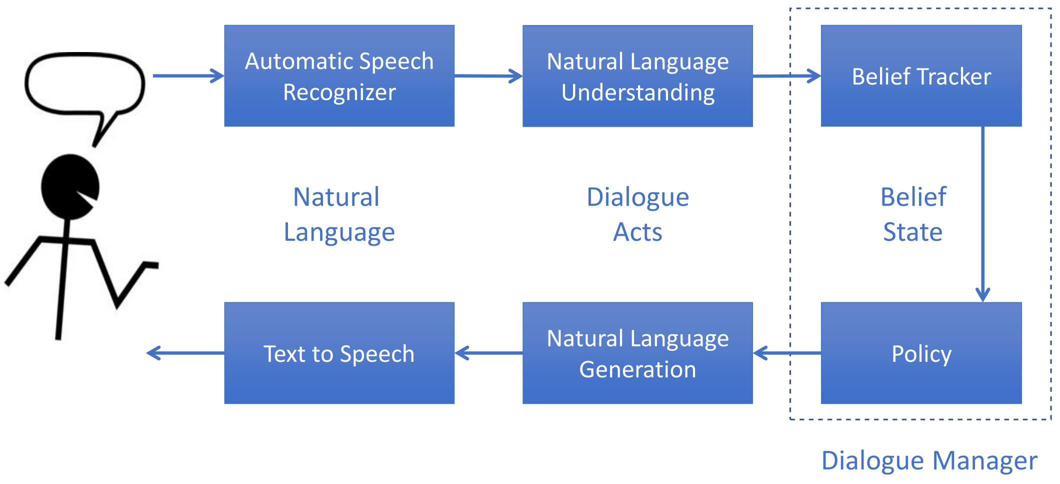

Figure 1.1: The main components of a Spoken Dialogue Systems (SDS)

The block diagram in Figure 1.1 represents the architecture of most Spoken Dialogue Systems. The Automatic Speech Recognizer (ASR) receives the spoken utterance from the user and outputs the most probable sequence of words. The recognized utterance is passed to a natural language understanding (NLU) module which decodes the utterance to form a semantic representation of the user input called a *user dialogue act* ("I would like Spanish food" $\rightarrow$ `inform(food=Spanish)`). The blocks succeeding the *Dialogue Manager* (DM) are used for the opposite task. The response of the system in its semantic representation, called *system dialogue act*, is converted into a sequence of words using a *Natural Language Generation* (NLG) component. This is followed by a text-to-speech (TTS) component that produces a speech waveform.

A task-oriented SDS is typically designed according to a structured *ontology*, which defines what the system can talk about. In a system recommending restaurants the ontology defines those attributes of a restaurant that the user can choose, called *informable slots* (e.g. different food types, areas and price ranges), the attributes that the user can request, called *requestable slots* (e.g. phone number or address) and the restaurants that it has data about. An attribute is referred to as a *slot* and has a corresponding *value*. Together these are referred to as a *slot-value pair* (e.g. `area`=north).

At the heart of an SDS sits the Dialogue Manager, which usually consists of two components. The first, called *belief tracker*, tries to infer the user goal from the dialogue history while maintaining good uncertainty estimates over the user goal. The uncertainty estimates are necessary due to erroneous ASR and NLU. The second is called the *policy*, which decides what to say given the output of the belief tracker and the NLU. Designing a policy that can respond appropriately in a task-oriented setting is non-trivial, particularly due to the omnipresence of speech recognition and language understanding errors leading to constant uncertainty about the true intention of the user. Traditionally, policies were based on hand-coded rules often in the form of flow charts. These systems are expensive to build, need extensive domain-specific knowledge and are often unable to recover from speech understanding errors (Young et al., 2013). A statistical, learned policy on the other hand can automatically learn dialogue rules and is often more robust to speech understanding errors. The uncertainty under which the DM takes decisions leads to policy learning being ill-suited for supervised learning, since it is not possible for an expert to label a data-set with perfect responses. This is because the uncertainty over the user goal that the expert experiences is different to the uncertainty that the model experiences. Thus, in state-of-the-art systems dialogue management is often formulated as a reinforcement learning (RL) problem. (Roy et al., 2000; Williams and Young, 2007; Gašić and Young, 2014). In this framework, the system learns through a *trial and error* process governed by a *reward function*. The reward function assigns a real-valued reward to each dialogue often in a way that encourages short dialogues that at the same time fulfil the user's goal. The

DM's policy is then trained to act such that it maximises the cumulative future reward.

1.1.2 Simulated training of a Dialogue System

Ideally, the DM's policy would be trained by interacting with real users. Although there are models that support on-line learning (Gašić et al., 2011), for the majority of RL algorithms, which require a lot of interactions, this is too expensive. Furthermore, a new set of users needs to be recruited every time a policy is trained. This makes common practices such as hyper-parameter optimization prohibitively expensive. Thus, it is natural to try to learn from a dataset which needs to be recorded only once, but can be used repeatedly. In theory, policy learning could be done offline using a recorded corpus of human-computer dialogue data and an off-policy RL algorithm. In practice however, the vast space of possible dialogue policies renders this approach impractical since the optimal policy is probably not contained within the dataset. A solution is to transform the static corpus into a dynamic tool: a *user simulator*. The user simulator (US) is trained on a dialogue corpus, collected using either a system prototype or a Wizard-of-Oz setup, to learn what responses a real user would provide in a given dialogue context. The US is trained using supervised learning since the aim is for it to learn *typical* user behaviour. For the DM, however, we want *optimal* behaviour which is why supervised learning cannot be used. By interacting with the SDS, the trained US can be used to train the DM. The DM is optimised using the feedback given by either the user simulator or more often a separate evaluator. Any number of dialogues can be generated using the US and dialogue strategies that are not in the recorded corpus can be explored. Afterwards, the DM can be fine-tuned through training in interaction with real users. This would be analogous to the separate pre-training and fine-tuning stages used in many areas of machine learning (Bahl et al., 1986; Povey and Woodland, 2002; Erhan et al., 2010).

Further, user simulators can also be used for just the evaluation of Spoken Dialogue Systems or as a model in model-based reinforcement learning. However, the focus of this report is building user simulators that can be used to train Spoken Dialogue Systems.

1.2 Motivation

Improving the user simulator used for training a DM is of paramount importance. Not only in order to improve the trained DM and thus to produce better consumer products, but also in order to strengthen research on reinforcement learning based dialogue managers. A large proportion of the research concerned with RL for dialogue management performs experiments with a US. Good performance using a poor user model can simply be the result of the DM exploiting weaknesses within the US. Using a better US can lead to more conclusive findings that are more likely to translate to experiments with real users.

Most proposed user simulators generate user responses at a semantic level rather than word level (Schatzmann et al., 2006). Multiple issues arise from this approach. Firstly, annotating the user-response with the correct semantics is costly. Larger datasets, given the same budget, could be collected if the US were to output natural language and thus not need semantic labels. Secondly, research suggests that the two modules of an SDS performing Natural Language Understanding (NLU) (inferring the meaning of the user utterance) and belief tracking (BT) (inferring the user goal from the dialogue history) should be jointly trained as a single entity (Mrkšić et al., 2017; Sun et al., 2016, 2014; Zilka and Jurcicek, 2015; Ramadan et al., 2018). In fact in the second Dialogue State Tracking Challenge (DSTC2) (Henderson et al., 2014), the data of which this project uses, the systems which only used joint systems for NLU and BT outperformed all systems that only used separate NLU and BT modules[1]. Training the policy of a DM in a simulated environment, when also using a joint system for NLU and belief tracking is *not possible* without a US that produces natural language. Thirdly, a US is sometimes augmented with an error model which generates a set of competing hypotheses with associated confidence scores trying to replicate the errors of the speech recogniser. When the error model matches the characteristics of the speech recogniser more accurately, the SDS performs better (Williams, 2008). However, speech recognition errors are badly modelled based on user semantics since they arise (mostly) due to the phonetics of the spoken words and not their semantics (Goldwater et al., 2010). Thus, an SDS that is trained with a natural language based error model is likely to outperform one trained with a semantic error model when tested on real users. User simulators with natural language as their output can be found in the literature. However, these are either a semantic US augmented by a natural language generator or US models that are not conditioned on any goal preventing it from being used to train a DM due to the impossibility of automatically determining if a dialogue was successful i.e. if the user goal was fulfilled.

Every user simulator needs some notion of a user state. The user state should capture information about the user's goal and the dialogue history, such that on its basis the next response of the user can be generated. The difficulty of the design of a good user simulator arises to a large extend from defining the user state. Mainly, due to a trade-off that has to be made between a user state that can be used to reproduce the complex characteristics of user behaviour while at the same time keeping the size of the state space small enough for the model to be trainable on limited amounts of data. A possible solution comes from learning not only the behaviour of the user simulator from data, but also the definition of what constitutes a good state. Deep Learning is based on the principle of *distributed representation*. This means that instead of defining a state-vector in which each dimension represents the presence or absence of a specific feature of the dialogue context,

[1]The best-performing models used both.

feature description is shared among all dimensions and each dimension participates in the representation of many concepts. The design of how the representation of concepts is shared among dimensions within the state-vector and the user behaviour in general can be learned directly from data, hence reducing necessary engineering efforts. This makes Deep Learning a prime candidate for building a good user simulator.

Multiple trainable user simulators have been proposed, but all those that do not use Deep Learning employ a designed notion of a user state. The currently most popular US for training a DM's policy is the Agenda-Based User-Simulator (Schatzmann et al., 2007). The mechanism that generates its user response is based on both a set of rules and a set of highly interpretable probabilities. Though these probabilities can be learned from data this is rarely done. Using rules leads to less diverse user behaviour which in turn leads to poor performance when testing with real users. Some Deep Learning based US models have been proposed though they either were not usable for training a DM's policy or modelled simplistic dialogues that do not include goal changes. Furthermore, a policy trained with a Deep Learning based US has yet to be evaluated either in interaction with human users or in interaction with a US that was not used for training.

1.3 Contributions

This project developed a Deep learning based user simulator termed Neural User Simulator (NUS). The NUS outputs natural language and its behaviour is learned from a corpus. The main component, inspired by El Asri et al. (2016), consists of a feature extractor and a neural network sequence-to-sequence model (Sutskever et al., 2014). The sequence-to-sequence model consists of a recurrent neural network (RNN) encoder that encodes the dialogue history into the user state and a decoder RNN which outputs natural language. Furthermore, the NUS generates its own goal and possibly changes it during a dialogue, which allows the model to be deployed for training more sophisticated DM policies.

The NUS is trained on dialogues between real users and an SDS in a restaurant recommendation domain. Compared to much of the related work on user simulation, we use the trained NUS to train the DM of a reinforcement learning based SDS. In order to evaluate the NUS, two further policies are trained. One using an Agenda-Based User-Simulator (ABUS) and another using an ABUS augmented with a deep learning based natural language generator. The three policies are compared against each other by using *cross-model* evaluation (Schatztmann et al., 2005). This means to train on one model and to test on the others. Furthermore, the two polices trained on the ABUS and the NUS are tested on real users recruited through Amazon Mechanical Turk. On both evaluation tasks, the NUS outperformed the other user simulators.

Chapter 2

Statistical User Modelling for Simulation-Based Dialogue Strategy Learning

2.1 Overview

Early research on User Modelling in the fields of Spoken Dialogue Systems (SDS) and Natural Language Processing (NLP) was mainly focused on building user models to represent individual users. This was to allow an SDS to adapt to a specific user. An approach analogous to humans using mental models of their dialogue partner in conversations.

The recent work on building user simulators to enable off-line training of the policy of dialogue managers differs in many respects from traditional work on user modelling in SDS and NLP. Primarily, it is no longer sufficient to represent an individual user since in this context user simulators should instead represent the entire diverse population of users. Therefore, a user simulator has to not just give a reasonable response in a specific dialogue context but also describe a probability distribution over responses. By sampling from this probability distribution, the collection of sampled responses should be representative of the user population as a whole. Thus, a statistical user simulator is needed. Such models also allow for the parameters of the models to be learned from a corpus of human-computer interactions. This is advantageous because the complexity of user behaviour prevents model parameters to be picked by hand. By learning the parameters from data, domain specific knowledge is not essential to building the system, which can lead to lower total development cost if the data collection is inexpensive enough.

Lastly, it is important to stress the importance of using an *explicit goal representation* for the user simulator. In order to use RL for DM policy optimization a reward has to be calculated. To calculate the reward it is necessary to know whether or not the user's goal has been fulfilled. If the user simulator is not conditioned on an explicit goal then the reward function cannot have access to the user's goal and hence cannot compare it to the system's responses in order to determine whether the dialogue was successful. In other cases lack of an explicit goal can lead to incoherent user behaviour throughout a dialogue. Another often forgotten factor is that within task-oriented dialogue most conceivable goals are not achievable e.g. it is likely that not every part of a town has a cheap, Ethiopian restaurant. In these cases, many users will change their goal. Thus, it is imperative for a user simulator to include a mechanism to change the goal in such events.

2.2 Intention Level User Modelling

Most user simulators operate at the level of user intentions, a compressed representation of human-computer interaction. User intentions are modelled by annotating user utterances with their semantics meanings. These cannot be directly observed but can be described using speech-act and dialogue-act theory (Searle, 1969; Bunt, 1981; Traum, 1999). User semantics usually consists of *user dialogue act* and a corresponding slot-value pair. Figure 2.1 shows the interaction of a US operating at the intention level with a DM augmented by an Error Model. The error model tries to replicate the errors of the speech recognizer. In publicly available implementations, error models on the semantic level randomly confuse either user dialogue act, the slot or the value. This can lead to absurd hypotheses such as `inform(addr)`. If the US is trainable then an expert annotator needs to label the real user response of the recorded corpus with the correct user semantics.

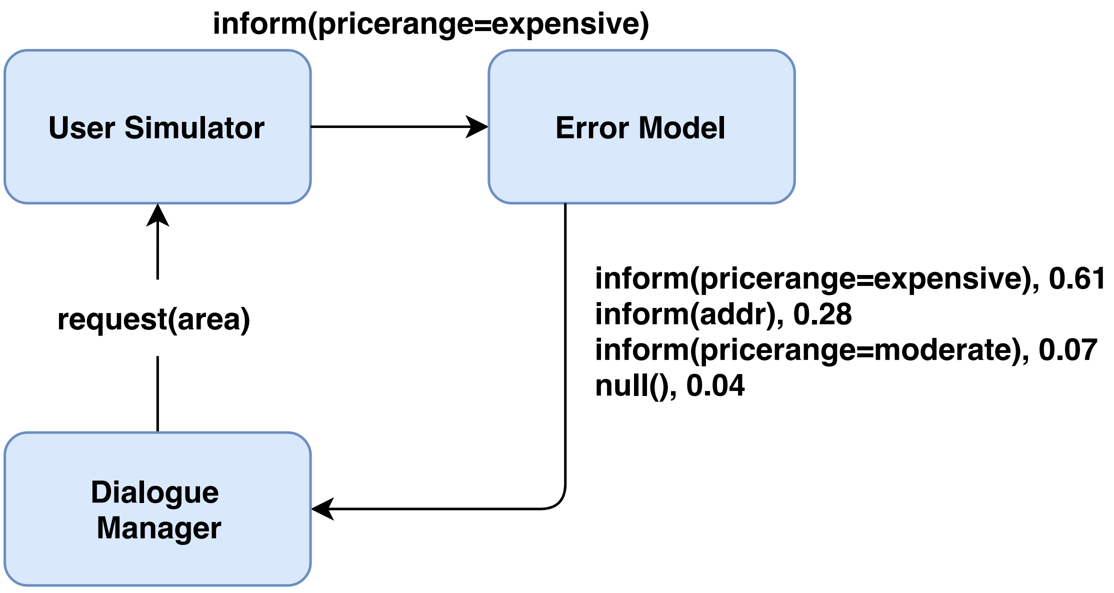
inform(pricerange=expensive)
User Simulator
Error Model
request(area)
inform(pricerange=expensive), 0.61
inform(addr), 0.28
inform(pricerange=moderate), 0.07
null(), 0.04
Dialogue Manager

Figure 2.1: Interaction between the Dialogue Manager and User Simulator at intention level.

The first statistical user simulator (Eckert et al., 1997) used a simple bi-gram model

$P(a_u \,|\, a_m)$ to predict the next user act a_u given the last system act a_m. It has the advantage of being purely probabilistic and domain-independent. However, it does not consider the full dialogue history and is not conditioned on a goal, leading to incoherent user behaviour throughout a dialogue. Scheffler and Young (2000, 2001) attempted to overcome goal inconsistency by proposing a graph-based model. The graph contains all possible "paths" that a user can take in a dialogue. Some nodes of the graph are probabilistic, for which the probabilities of each choice can be estimated from data. However, developing the graph structure requires extensive domain-specific knowledge and large engineering efforts. Pietquin and Dutoit (2006) combined features from Sheffler and Young's work with Eckert's Model, by conditioning a set of probabilities on an explicit representation of the user goal and memory. Even though the parameters of this model can be learned from data, Pietquin hand-selected all model parameters for his work. A Markov Model is also used by Georgila et al. (2005). It uses a large feature vector to describe the user's current state, which helps to compensate for the Markov assumption. However, the model is not conditioned on any goal. Therefore, it is not used to train a dialogue policy since it is impossible to determine whether the user goal was fulfilled. A hidden Markov model was proposed by Cuayáhuitl et al. (2005), which was also never used to train a policy. Chandramohan et al. (2011) cast user simulation as an inverse reinforcement learning problem where the user is modelled as a decision-making agent. Their model did not incorporate a user goal and was hence not used to train a policy. The most prominent user model for policy optimisation is the Agenda-Based User Simulator (ABUS) (Schatzmann et al., 2007), which represents the user state elegantly as a stack of necessary user dialogue acts called the *agenda*. The agenda is ordered according to the priorities of each of the user dialogue acts. For each dialogue turn, the ABUS updates the agenda based on the system's response and then pops a certain number of user dialogue acts off the agenda which then constitute the user's response. The model is explicitly conditioned on a goal which can be updated in case the dialogue system expresses that it cannot fulfil the goal. The updating of the agenda and the change of the goal is based on a small set of highly interpretable probabilities. This allows for the probabilities (model parameters) to be set manually. Therefore the ABUS can be used to train an SDS without the need for suitable data of human-machine interactions. Furthermore, model parameters can be set in order to encourage certain types of behaviour, such as not providing much information unless prompted. This change in user behaviour enables interesting experiments to be done for the advancement of RL research. Although the model parameters of the ABUS can be learned from data, as done by Keizer et al. (2010), this is rarely done and most publicly available implementation are based on hand-picked parameters. El Asri et al. (2016) modelled user simulation as a sequence-to-sequence task modelled by a neural network, where the output sequence is a sequence of user dialogue acts. Therefore, the model can keep track of the dialogue history and user behaviour is learned entirely from data. However, goal changes were not modelled, even though a large proportion of dialogues within their dataset (DSTC2) contains goal changes. Their model did outperform the ABUS on statistical metrics of similarity between the US output and real user outputs within a test set. However, this is not surprising given that it was trained by optimising a similar statistical metric and the ABUS was not. The ABUS was specifically proposed for the training of an SDS which is what a metric used for evaluation should be based on.

2.3 Word Level User-Modelling

Word-Level User Modelling, building a US that generates natural language, is performed by far rarer than Intention level User Modelling and up until recently it was thought that a Word-Level US cannot be robust enough and scalable enough (Schatzmann et al., 2006; Schatzmann, 2008). Early research in this area was essentially limited to two publications by the same group of authors, both focusing on dialogue strategy evaluation rather than learning (Araki et al., 1997; Watanabe et al., 1998). Figure 2.2 shows how a Word-Level User Simulator that uses the system's output in its semantic form would be interfaced to train the DM's policy. The error model could now base its confusions on the spoken words. If the `t` of 'want' is not properly pronounced, 'want' could be confused with 'one'. Similarly, 'an expensive' is easily confusable with 'inexpensive'. It is also clear to see how now a Dialogue Manager could be used whose Belief Tracker performs NLU at the same time. Furthermore, given that the US now directly outputs an utterance of words, labels of the semantic meaning of a user's utterance are not needed for training the US.

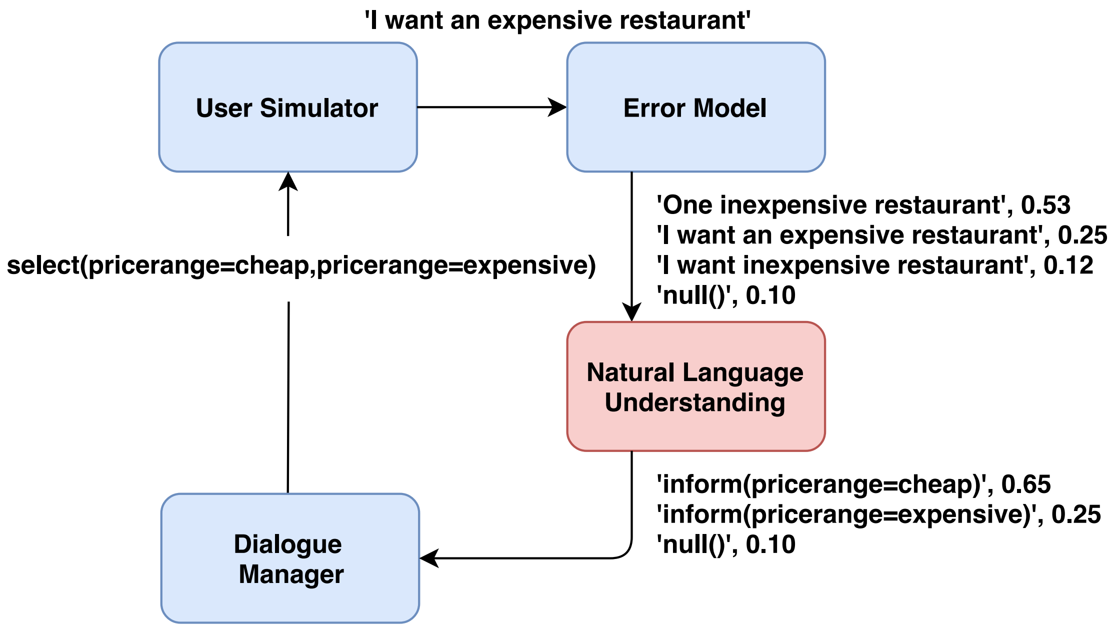
'I want an expensive restaurant'
User Simulator
Error Model
'One inexpensive restaurant', 0.53
'I want an expensive restaurant', 0.25
'I want inexpensive restaurant', 0.12
'null()', 0.10
select(pricerange=cheap,pricerange=expensive)
Natural Language Understanding
'inform(pricerange=cheap)', 0.65
'inform(pricerange=expensive)', 0.25
'null()', 0.10
Dialogue Manager

Figure 2.2: Interaction between the Dialogue Manager and User Simulator at word level.

Sequence-to-sequence learning for word-level user simulation is performed in (Crook and Marin, 2017), though the model is not conditioned on any goal and hence not used for policy optimisation. Furthermore, in comparison to the intention of this project the input to their US is also text. Thus, the US not only has to learn the already hard task of natural language generation, but also the task of natural language understanding. Therefore such model is likely to require by far more data than a model using the system semantics as its input in order to model the user to a reasonable degree.

A word-level user simulator was also used in (Liu and Lane, 2017) where a neural sequence-to-many model was used to model the user on the intention level, which was then augmented by a template based natural language generator (NLG). Li et al. (2017) augmented the ABUS with a learned natural language generator to obtain a word-level user simulator. The latter approach will also be used for comparison in this project, but with a more advanced NLG, the Semantically-Conditioned LSTM developed by Wen et al. (2015).

Chapter 3

Deep Learning

Deep Learning is a subfield of Machine Learning. Deep Learning concerns designing and training *Artificial Neural Networks* (ANNs). An ANN defines a function that maps an input vector $\mathbf{x}$ to an output vector $\mathbf{y}$. Generally, the ANN will be defined through multiple steps of computation, called *layers*. The input to a specific layer will be a vector and the output of this layer will be another vector that is passed to other layers (or in the case of a Recurrent Neural Network back to the same layer, Sec. 3.3). There is no agreed upon definition of a layer, but it generally consists of differentiable operations, usually matrix-multiplications, element-wise multiplications, and differentiable functions operating on one or multiple entries. Due to the ANN being differentiable, its parameters can be optimized, or rather learned, using gradient-based methods. The entire ANN will now consist of multiple layers assembling a computational graph with nodes building upon other nodes. Due to the computational graph having a plethora of nodes in sequential order, this approach to machine learning is termed *Deep* Learning.

3.1 Feedforward Neural Networks

In Feedforward Neural Networks, the computational graph is acyclic, meaning that the output from one layer is passed to succeeding layers and not any preceding layer or back to itself. The simplest form of layer is a fully connected layer defined as:

$$\mathbf{h^1} = f(W^1\mathbf{x} + \mathbf{b^1}) = f(\mathbf{a^1}) \tag{3.1}$$

where W^1 and $\mathbf{b^1}$ are termed the weight-matrix and bias vector, whose parameters will be learned during training. $\mathbf{x}$ and $\mathbf{h^1}$ are input and output to the layer. f, called the activation function, is any suitable non-linear function. Usually, pointwise functions are

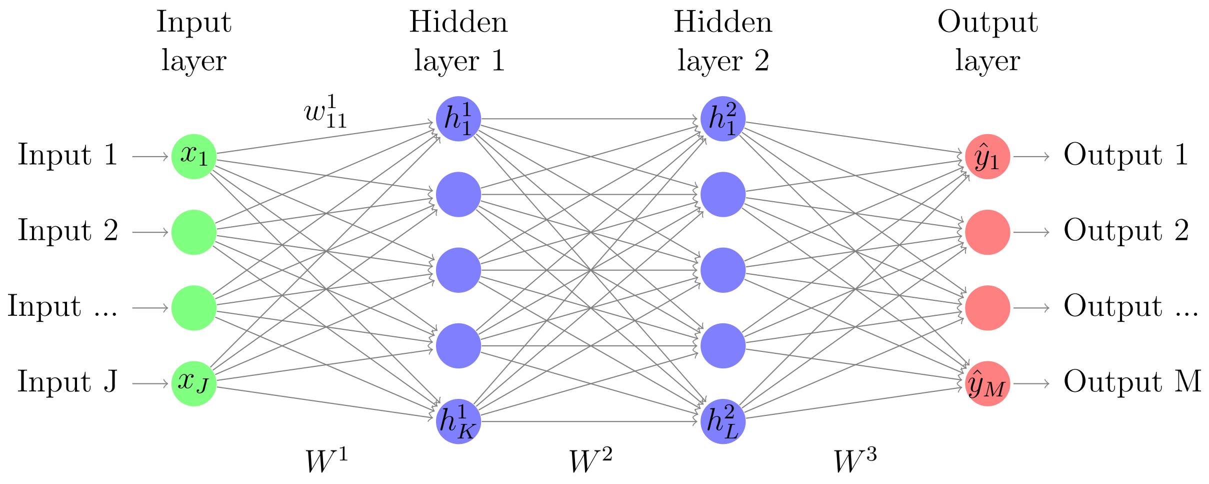

Figure 3.1: A fully-connected feedforward Artificial Neural Network (ANN) with two hidden-layers. Bias vectors are omitted for clarity.

used such as the logistic-sigmoid, $\sigma(\cdot)$, or the hyperbolic tangent, $\texttt{tanh}$, given by:

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

$$\texttt{tanh}(x) = \frac{e^{2x}-1}{e^{2x}+1} \tag{3.3}$$

In a *deep* fully connected ANN, equation 3.1 is repeatedly applied.

$$\mathbf{h^i} = f(W^i\mathbf{h^{i-1}} + \mathbf{b^i}) = f(\mathbf{a^i}) \tag{3.4}$$

This is illustrated in Figure 3.1, where a deep fully-connected ANN with two hidden layers is shown. Each drawn edge corresponds to an entry within the weigh-matrix and the bias vector is omitted for clarity. For a classification task the Output layer would be designed to be a probability distribution over M classes. This is achieved by passing the output of the Hidden layer 2 through an affine transform followed by the softmax activation function. The results of the affine transform, $\mathbf{z}$, will be referred to as *logits*.

$$\mathbf{z} = W^3\mathbf{h^2} + \mathbf{b^3} \tag{3.5}$$

$$p(\text{class} = i \mid \mathbf{x}) \;\hat{=}\; \hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{M} e^{z_j}} \tag{3.6}$$

In a training set used for a classification task we would have corresponding pairs between the input $\mathbf{x}$ and the correct output $\mathbf{y}$, were $\mathbf{y}$ would be a 1-hot vector where the index corresponding to the correct class is set to 1. There are no known correct values for $\mathbf{h^1}$ and $\mathbf{h^2}$, which is where the name *hidden layer* originates from.

3.2 Parameter Optimization

For any ANN we need to determine suitable parameters $\boldsymbol{\theta}$ (i.e. weight matrices and bias vectors). This is done by choosing a suitable cost function $C(\boldsymbol{\theta})$ and altering $\boldsymbol{\theta}$ to reduce $C(\boldsymbol{\theta})$. The simplest algorithm to optimize $C(\boldsymbol{\theta})$ is called *Gradient Descent*. It involves finding the gradient of C with respect to (w.r.t.) $\boldsymbol{\theta}$, $\frac{\partial C}{\partial \theta}$, and then changing the weights in the opposite direction of the gradient:

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{\text{old}} - \eta \left. \frac{\partial C}{\partial \boldsymbol{\theta}} \right|_{\boldsymbol{\theta}_{\text{old}}} \tag{3.7}$$

where η is called the *learning rate*.

3.2.1 Cross-Entropy Loss

As mentioned before, in classification tasks the softmax activation function is used and its output interpreted as a probability distribution over the possible classes that the ANN assigns to the input $\mathbf{x}$. An intuitive objective is to train the ANN such that it assigns the highest probability possible to the N training samples $\{\mathbf{x}^i, \mathbf{y}^i\}_{i=1}^N$, termed *Maximum Likelihood* training.

$$\begin{aligned}
\boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{N} p(\mathbf{y}^i \mid \mathbf{x}^i) && (3.8)\\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} -\sum_{i=1}^{N} \log p(\mathbf{y}^i \mid \mathbf{x}^i) && (3.9)\\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} -\sum_{i=1}^{N} \sum_{k=1}^{M} y_k^j \log p(\mathbf{y}^i \mid \mathbf{x}^i) && (3.10)\\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} -\sum_{i=1}^{N} \sum_{k=1}^{M} y_k^i \log \hat{y}_k^i && (3.11)\\
\therefore C(\boldsymbol{\theta}) &= -\sum_{i=1}^{N} \sum_{k=1}^{M} y_k^i \log \hat{y}_k^i && (3.12)
\end{aligned}$$

This cost function is most-used in classification and usually referred to by its name from Information Theory called *Cross-Entropy*, though it is more intuitive to derive it through the Maximum-Likelihood framework.

In the following we will consider gradients of C for a single sample only ($N = 1$). To obtain these gradients for datasets with $N \neq 1$ one would have to simply sum these equations over all samples. The cost-function from equation 3.12 has another useful property which is that the gradient of C w.r.t. the logits ($\mathbf{z}$, see equation 3.5), for a single data point, has a simple closed form expression given by:

$$\frac{\partial C}{\partial z_k} = \hat{y}_k - y_k \tag{3.13}$$

3.2.2 Back-propagation

Given the gradient of the C w.r.t. the linear activations $\mathbf{a}^i$ of hidden layer i $\frac{\partial C}{\partial \mathbf{a}^i}$, finding the gradients w.r.t the parameters of the layer is trivial:

$$\frac{\partial C}{\partial W^i} = \frac{\partial C}{\partial \mathbf{a}^i}\ \mathbf{h}^{i-1T} \tag{3.14}$$

$$\frac{\partial C}{\partial \mathbf{b}^i} = \frac{\partial C}{\partial \mathbf{a}^i} \tag{3.15}$$

The challenge comes when trying to use $\frac{\partial C}{\partial \mathbf{a}^i}$ to find $\frac{\partial C}{\partial \mathbf{a}^{i-1}}$. This would be to *propagate* the gradients *back* from one layer to an earlier layer, which is where the term *back-propagation* originates from. This can be achieved by applying the chain rule of calculus:

$$\frac{\partial C}{\partial \mathbf{a}^{i-1}} = \frac{\partial \mathbf{a}^i}{\partial \mathbf{a}^{i-1}}\frac{\partial C}{\partial \mathbf{a}^i} \tag{3.16}$$

$$= \frac{\partial \mathbf{h}^{i-1}}{\partial \mathbf{a}^{i-1}}\frac{\partial \mathbf{a}^i}{\partial \mathbf{h}^{i-1}}\frac{\partial C}{\partial \mathbf{a}^i} \tag{3.17}$$

$$\frac{\partial a^i_n}{\partial h^{i-1}_m} = \left[W^i\right]_{nm} = \left[W^{iT}\right]_{mn} \tag{3.18}$$

$$\therefore \frac{\partial C}{\partial \mathbf{a}^{i-1}} = \frac{\partial \mathbf{h}^{i-1}}{\partial \mathbf{a}^{i-1}} W^{iT} \frac{\partial C}{\partial \mathbf{a}^i} \tag{3.19}$$

This process is initialized using the gradients from equation 3.13.

$\frac{\partial \mathbf{h}^{i-1}}{\partial \mathbf{a}^{i-1}}$ will depend on the particular activation function used. In most cases, element-wise activation functions will be used and thus $\frac{\partial \mathbf{h}^{i-1}}{\partial \mathbf{a}^{i-1}}$ will be diagonal. For the sigmoid activation function:

$$\frac{d\sigma\left(x\right)}{dx} = \sigma\left(x\right)\left(1 - \sigma(x)\right) \tag{3.20}$$

$$\therefore \frac{\partial \mathbf{h}^{i-1}}{\partial \mathbf{a}^{i-1}} = \texttt{diag}\left(\sigma\left(\mathbf{a}^{i-1}\right)\right)\texttt{diag}\left(\mathbf{1} - \sigma\left(\mathbf{a}^{i-1}\right)\right) \tag{3.21}$$

Looking at equation 3.19 one can see that if the spectral radius of $\frac{\partial \mathbf{h}^{i-1}}{\partial \mathbf{a}^{i-1}} W^{iT}$ is above 1, then the gradients will become ever larger as we propagate the gradients back to earlier layers. If the spectral radius is below one then the gradients will become ever smaller. These two problems are termed *exploding* and *vanishing* gradients. The gradient of sigmoid function, $\sigma(.)$, is always below 0.25. Furthermore, regularization techniques are sometimes used, which try to limit the sizes of the individual weights within a weight-matrix which would lead to the spectral radius of W^{iT} being small. Therefore, *vanishing* gradients are more often a problem within Deep Learning.

3.3 Recurrent Neural Networks

In a Recurrent Neural Network (RNN), the computational graph is cyclic, meaning that the output of a layer is fed into either the same or a preceding layer at the next time-step. It is a family of ANNs primarily for processing sequential data $\mathbf{x_1}, ..., \mathbf{x_T}$. An RNN can be defined as:

$$(\mathbf{h}_t, \mathbf{s}_t) = \mathrm{RNN}\left(\mathbf{x}_t, \mathbf{s}_{t-1}\right) \tag{3.22}$$

At time-step t, an RNN uses an input $\mathbf{x}_t$ and an internal state $\mathbf{s}_{t-1}$ to produce its output $\mathbf{h}_t$ and its new internal state $\mathbf{s}_t$. This very general definition allows not only for many different RNN version (Hochreiter and Schmidhuber, 1997; Cho et al., 2014) for standard sequences, but also allows RNNs to be arranged in more complex ways (Kalchbrenner et al., 2016) and combined with other types of neural networks (Kreyssig et al., 2018b). The fully-connected layer of equation 3.1 can be extended to an RNN, by concatenating the previous output of the layer with the current input:

$$\mathbf{h^1_t} = f\left(W^1 \begin{bmatrix} \mathbf{x_t} \\ \mathbf{h^1_{t-1}} \end{bmatrix} + \mathbf{b^1}\right) \tag{3.23}$$

where $\mathbf{h^1_t}$ would be both the output and the internal state.

3.3.1 Long Short-Term Memory Networks

Since the output of the recurrent layer is passed back to its input at the next time-step an RNN can be seen as a potentially infinitely deep ANN. Therefore RNNs suffer severely from the problem of vanishing and exploding gradients.

When the $\sigma(\cdot)$ activation function is used, its mainly vanishing gradients that occur. This problem was addressed by Hochreiter and Schmidhuber (1997) through introducing the Long Short-Term Memory RNN (Hochreiter and Schmidhuber, 1997), defined by:

$$\mathbf{\tilde{c}_t} = tanh(W_c\mathbf{x_t} + U_c\mathbf{h_{t-1}} + \mathbf{b_c}) \tag{3.24}$$

$$\mathbf{i_t} = sigm(W_i\mathbf{x_t} + U_i\mathbf{h_{t-1}} + \mathbf{b_i}) \tag{3.25}$$

$$\mathbf{f_t} = sigm(W_f\mathbf{x_t} + U_f\mathbf{h_{t-1}} + \mathbf{b_f}) \tag{3.26}$$

$$\mathbf{c_t} = \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \mathbf{\tilde{c}_t} \tag{3.27}$$

$$\mathbf{o_t} = sigm(W_o\mathbf{x_t} + U_o\mathbf{h_{t-1}} + \mathbf{b_o}) \tag{3.28}$$

$$\mathbf{h_t} = \mathbf{o_t} \odot tanh(\mathbf{c_t}) \tag{3.29}$$

the element-wise multiplications ($\odot$) and especially the addition operation in Equation 3.27 lead to less decreased gradients when calculating $\frac{\partial C}{\partial \mathbf{c_{t-1}}}$ based on $\frac{\partial C}{\partial \mathbf{c_t}}$.

3.3.2 Natural Language Generation using SC-LSTM

As previously mentioned in Section 2.3 a recurrent neural network can be used to generate a user response at the word level from its semantic form. This section describes how this is done using the semantically conditioned LSTM (SC-LSTM) introduced by Wen et al. (2015).

The model is based on an LSTM language model in which a 1-hot encoding $\mathbf{w}_{t-1}$ of the previous word w_{t-1} is input at each time step t $\mathbf{x_t} = \mathbf{w}_{t-1}$. The output of the neural network is then a probability distribution over the next word w_t. The probability distribution is obtained by extending the output of the LSTM (Equation 3.29) with an affine transform and the softmax activation function as in Equations 3.5 and 3.6. Therefore, by sampling words one by one from the output distribution of the RNN until an end of sentence token (`<EOS>`) is generated, the network can produce a sequence of words. In our case we would like the output to be independent to the particular slot-value i.e. the output sentence for inform(`food`=Spanish) should also be usable for inform(`food`=Ethiopian). This is done by replacing all possible values for a specific slot with a slot-token (e.g. `SLOT_FOOD`) within the training data, a process called *delexicalisation*. The Language Generation is then followed by a post-processing step called *lexicalisation* in which the slot-tokens are replaced by their slot-values. This process allows the model to generalize to previously unseen slot-values.

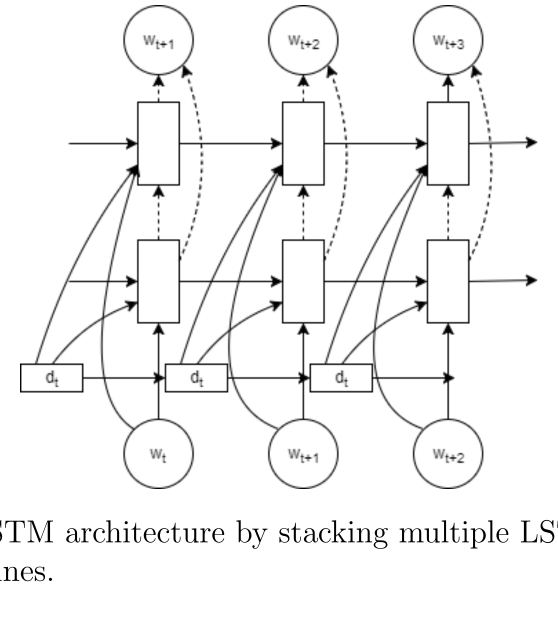

Figure 3.2: Deep SC-LSTM architecture by stacking multiple LSTM layers. Dropout was applied to the dashed lines.

In order to ensure that the generated utterance represents the intended meaning, the generator is further conditioned on a control vector $\mathbf{d}_0$, a binary representation of the

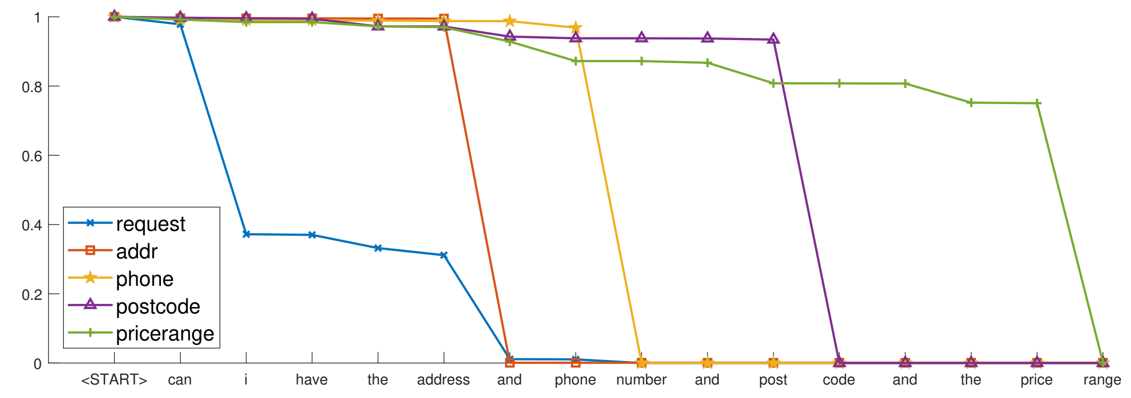

Figure 3.3: Values of the dialogue act cell when producing a sentence for `request(addr,postcode,phone,pricerange)`

dialogue act type and its slot-value pairs. In order not to duplicate slot information in the generated utterance an additional control cell, $\mathbf{r}_t$ called the reading gate, is introduced to modulate $\mathbf{d}_t$. Starting from the original dialogue act binary vector $\mathbf{d}_0$, at each time step the reading gate decides what information should be retained for future time step.

$$\mathbf{r}_t = \sigma\left(W_{wr}\mathbf{w}_t + \alpha W_{hr}\mathbf{h}_{t-1}\right) \tag{3.30}$$

$$\mathbf{d}_t = \mathbf{r}_t \odot \mathbf{d}_{t-1} \tag{3.31}$$

Here W_{wr} and W_{hr} act like keyword and key phrase detectors that learn to associate certain patterns of generated words with certain slots. Figure 3.3 gives an example of how these detectors work in affecting $\mathbf{d_t}$. Due to the way in which $\mathbf{d}_0$ is set, $\mathbf{d_t}$ is highly interpretable and specific dimensions of it correspond to specific dialogue acts or specific slots. In Figure 3.3 specific dimensions of $\mathbf{d_t}$ are plotted as the SC-LSTM generates a sentence for the user requesting the values of four slots of a restaurant from the SDS. As desired, after a word is generated related to a specific slot or dialogue act, the corresponding value in $\mathbf{d}_t$ goes to zero. For example, after 'address' is emitted the dimension which we set to `1` in $\mathbf{d}_0$ every time 'address' is part of the semantics goes to 0 in $\mathbf{d}_t$.

In the dataset no more than three slots are ever requested, but in Fig. 3.3 four slots are requested. This shows that the model is able to generalize very well to unseen user semantics.

To condition the network on $\mathbf{d_t}$ Equation 3.27 is modified so that cell value $\mathbf{c}_t$ depends on $\mathbf{d_t}$,

$$\mathbf{c_t} = \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \tilde{\mathbf{c}}_\mathbf{t} + tanh(W_{dc}\mathbf{d}_t) \tag{3.32}$$

Wen et al. (2015) found that stacking multiple SC-LSTM layers give improvements. This is done, by using the output of the first layer as the input to the next ($\mathbf{x_t^2} = \mathbf{h_t^1}$) and modifying Equation 3.30 so that all hidden information can influence the reading gate:

$$\mathbf{r}_t = \sigma\left(W_{wr}\mathbf{w}_t + \sum_l \alpha_l W_{hr}^l \mathbf{h}_{t-1}^l\right) \tag{3.33}$$

where l is the hidden layer index and α_l is a layer-wise constant.

Furthermore, skip connections, from the input w_t to all hidden layers, and from all hidden layers to the output were used (see Figure 3.2 for an illustration) as in (Wen et al., 2015).

Chapter 4

Neural User Simulator

An overview of the Neural User Simulator (NUS) is given in Figure 4.1. At the start of a dialogue the *Goal Generator* generates a random goal G_0. The possibilities for G_0 are defined by the *ontology.* In dialogue turn T, the output of the SDS (da_T) is passed to the NUS's *Feature Extractor*, which generates a feature vector $\mathbf{v}_T$ based on da_T, the current user goal, G_T, and parts of the dialogue history. This vector is appended to the *Feature History* $\mathbf{v}_{1:T} = \mathbf{v}_1 ... \mathbf{v}_T$. This sequence is passed to the *sequence-to-sequence* model (Fig. 4.2), which will generate the user's length n_T utterance $\mathbf{u}_T = w_0 ... w_{n_T}$. As in Figure 4.2, words in $\mathbf{u}_T$ corresponding to a slot are replaced by a slot token; a process called *delexicalisation.* If the SDS expresses to the NUS that there is no venue matching the NUS's constraints, the the Goal Generator will alter the goal.

4.1 Goal Generator

The Goal Generator generates a random goal $G_0 = (C_0, R)$ at the start of the dialogue. It consists of a set of *constraints*, C_0, which specify the required venue e.g. (`food`=Spanish, `area`=north) and a number of *requests*, R, that specify the information that the NUS wants about the final venue e.g. the address or the phone number. The possibilities for C_t and R are defined by the *ontology.* In DSTC2 C_t can consist of a maximum of three constraints; `food`, `area` and `pricerange`. Whether each of the three is present is independently sampled with a probability of 0.66, 0.62 and 0.58 respectively. These probabilities were estimated from the DSTC2 data set. If no constraint is sampled then the goal is re-sampled. For each slot in C_0 a value (e.g. north for `area`) is sampled uniformly from the ontology. Similarly, the presence of a request is independently sampled, followed by re-sampling if zero requests were chosen.

When training the sequence-to-sequence model, the Goal Generator is not used, but instead the goal labels from the DSTC2 dataset are used. In DSTC2 one goal-label is given to the entire dialogue. This goal is always the *final* goal. If the user's goal at the

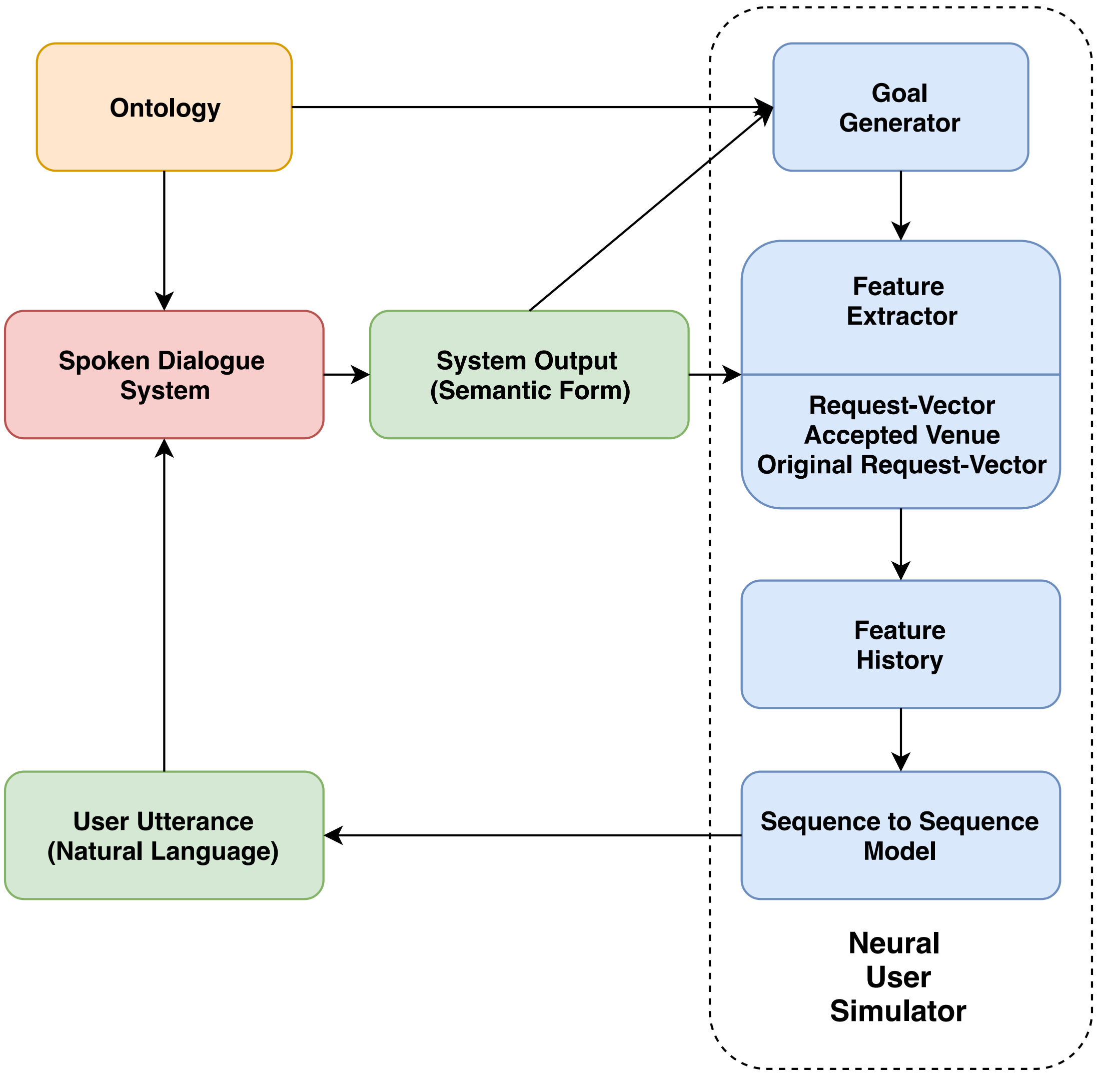

Figure 4.1: General Architecture of the Neural User Simulator and its interface with the Spoken Dialogue System (SDS). The System Output is passed, in its semantic form, to the Feature Extractor. It generates a new feature vector that is appended to the Feature History, which is passed to the sequence-to-sequence model to produce the user utterance. At the start of the dialogue, the Goal Generator generates a goal, which might change during the course of the dialogue based on the system output. The scope of both the goal generator and the SDS is defined by the ontology.

start of the dialogue is (`food`=eritrean, `area`=south), which is changed to (`food`=spanish, `area`=south), due to the non-existence of an Eritrean restaurant in the south, using only the final goal is *insufficient* to model the dialogue. The final goal can only be used for the requests, as they are not altered during a dialogue. DSTC2 also provides turn-specific labels. These contain the constraints and requests expressed by the user up until and including the current turn. When training a policy with the NUS, such labels would not be available as they "predict the future", i.e. when the turn-specific constraints change from (`area`=south) to (`food`=eritrean, `area`=south) it means that the user will inform the system about her desire to eat Eritrean food in the current turn.

4.1.1 Goal-Labels in DSTC2

C_t	**Original**	**Updated**
C_0	food=eritrean	area=south, food=eritrean, pricerange=cheap
C_1	area=south, food=eritrean	area=south, food=eritrean, pricerange=cheap
C_2	area=south, food=spanish	area=south, food=spanish, pricerange=cheap
C_3	area=south, food=spanish, pricerange=cheap	area=south, food=spanish, pricerange=cheap

Table 4.1: An example of how DSTC2's turn-specific constraint labels can be transformed such that their behaviour can be replicated when training a dialogue manager.

In related work on user-simulation for which the DSTC2 dataset was used, the final goal was used for the entire dialogue (El Asri et al., 2016; Serras et al., 2017; Liu and Lane, 2017). As stated above, we do not believe this to be sufficient. The following describes how to update the turn-specific constraint labels such that their behaviour can be replicated when training a DM's policy, whilst allowing goal changes to be modelled. The update strategy is illustrated in Table 4.1 with an example. The final turn keeps its constraints, from which we iterate *backwards* through the list of DSTC2's turn-specific constraints. The constraints of a turn will be set to the *updated* constraints of the succeeding turn, besides if the same slot is present with a *different* value. In that case the value will be kept. The behaviour of the updated turn-specific goal-labels can be replicated when the NUS is used to train a DM's policy. In the example, the food type changed due to the SDS expressing that there is no restaurant serving Eritrean food in the south. When deploying the NUS to train a policy, the goal is updated when the SDS outputs the `canthelp` dialogue act.

4.2 Feature Extractor

The Feature Extractor generates the feature vector that is appended to the sequence of feature vectors, here called *Feature History*, that is passed to the sequence-to-sequence model. The input to the Feature Extractor is the output of the DM and the current goal G_t. Furthermore, as indicated in Figure 4.1, the Feature Extractor keeps track of the currently accepted venue as well as the current and initial *request-vector*, which is explained below.

The feature vector $\mathbf{v}_t = [\mathbf{a}_t\, \mathbf{r}_t\, \mathbf{i}_t\, \mathbf{c}_t]$ is made up of four sub-vectors. The motivation behind the way in which these four vectors were designed is to provide an embedding for the system response that preserves all necessary *value-independent* information. The use of such embedding is inspired by El Asri et al. (2016), but the specific method is different mainly due to the fact that an word-level US needs more information than an intention-level US. The US of (El Asri et al., 2016) would respond to `request(food)` with `inform` and a post-processing step would convert this into `inform(food=...)`. The sentence structure of a user's response to `request(food)` and `request(area)` are very different. This does not have to be considered for an intention-level US, therefore the embeddings from (El Asri et al., 2016) could not be used for this work.

The first vector, *machine-act vector* $\mathbf{a}_t$, encodes the dialogue acts of the system response and consists of two parts; $\mathbf{a}_t = [\mathbf{a}_t^1\ \mathbf{a}_t^2]$. $\mathbf{a}_t^1$ is a binary representation of the system dialogue acts present in the input. Its length is thus the number of possible system dialogue acts. It is binary and not one-hot since in DSTC2 multiple dialogue acts can be in the system's response. $\mathbf{a}_t^2$ is a binary representation of the slot if the dialogue act is `request` or `select` and if it is `inform` or `expl-conf` together with a *correct* slot-value pair for an informable slot. The length is four times the number of informable slots. $\mathbf{a}_t^2$ is necessary due to the dependence of the sentence structure on the exact slot mentioned by the system.

The second vector, *request-vector* $\mathbf{r}_t$, is a binary representation of the requests that have not yet been fulfilled. It's length is thus the number of requestable slots. In comparison to the other three vectors, the feature extractor needs to remember it for the next turn. At the start of the dialogue, the indices corresponding to requests that are in R are set to 1 and the rest to 0. Whenever the system informs about a certain request the corresponding index in $\mathbf{r}_t$ is set to 0. When a new venue is proposed, $\mathbf{r}_t$ is reset to the original request vector, which is why the Feature Extractor keeps track of it.

The third vector, *inconsistency-vector* $\mathbf{i}_t$, represents the inconsistency between the system's response and C_t. Every time a slot is mentioned by the system, when describing a venue (`inform`) or confirming a slot-value pair (`expl-conf` or `impl-conf`), the indices corresponding to the slots that have been misunderstood are set to 1. The length of $\mathbf{i}_t$ is the number of informable slots. This vector is necessary in order for the NUS to correct the system.

The fourth vector, $\mathbf{c}_t$, is a binary representation of the slots that are in the constraints C_t. It's length is thus the number of informable slots. This vector is necessary in order for the NUS to be able to inform about its preferred venue.

4.3 Sequence-To-Sequence Model

The sequence-to-sequence model (Figure 4.2) consists of an RNN encoder, followed by a fully-connect layer and an RNN decoder. As described in Section 3.3 an RNN can be defined as:

$$(\mathbf{h}_t, \mathbf{s}_t) = \mathrm{RNN}\left(\mathbf{x}_t, \mathbf{s}_{t-1}\right) \tag{4.1}$$

At time-step t, an RNN uses an input $\mathbf{x}_t$ and an internal state $\mathbf{s}_{t-1}$ to produce its output $\mathbf{h}_t$ and its new internal state $\mathbf{s}_t$. There is a plethora of different RNN architectures that

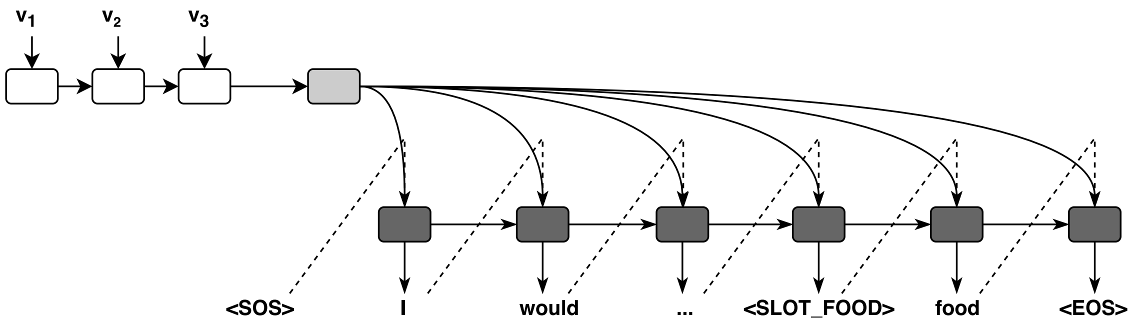

Figure 4.2: Sequence-To-Sequence model of the Neural User Simulator. Here, the NUS is generating the user response to the third system output. The white, light-grey and dark blocks represent the RNN encoder, a fully connected layer and the RNN decoder respectively. The previous output of the decoder is passed to its input for the next time-step. $\mathbf{v}_{3:1}$ are the first three feature vectors (see Sec. 4.2).

could be used and explored. Given that such exploration is not the focus of this work a single layer LSTM (Hochreiter and Schmidhuber, 1997) is used for both the RNN encoder and decoder.

The first RNN (shown as white blocks in Fig. 4.2) takes one feature vector $\mathbf{v}_t$ at a time as its input ($\mathbf{x}_t^E = \mathbf{v}_t$). If the current dialogue turn is turn T then the final output of the RNN encoder is given by $\mathbf{h}_T^E$, which is passed through a fully-connected layer (shown as the light-grey block) with linear activation function:

$$\mathbf{p}_T = W_p \mathbf{h}_T^E + \mathbf{b}_p \tag{4.2}$$

For a certain encoding $\mathbf{p}_T$ the sequence-to-sequence model should define a probability distribution over different sequences. By sampling from this distribution, the NUS can generate a diverse set of sentences corresponding to the same dialogue context. The conditional probability distribution of a length L sequence is defined as:

$$P(\mathbf{u}\,|\,\mathbf{p}) = P(w_0\,|\,\mathbf{p}) \prod_{t=1}^{L} P(w_t\,|\,w_{t-1}...w_0, \mathbf{p}) \tag{4.3}$$

The decoder RNN (shown as dark blocks) will be used to model $P(w_t\,|\,w_{t-1}...w_0, \mathbf{p})$. It's input at each time-step is the concatenation of an embedding $\mathbf{w}_{t-1}$ (we used 1-hot) of the previous word w_{t-1} ($\mathbf{x}_t^D = [\mathbf{w}_{t-1}\ \mathbf{p}]$). For $P(w_0\,|\,\mathbf{p})$ a *start-of-sentence* (`<SOS>`) token is used as w_{-1}. The end of the utterance is modelled using an *end-of-sentence* (`<EOS>`) token. When the decoder RNN generates the *end-of-sentence* token, the decoding process is terminated. The output of the decoder RNN, $\mathbf{h}_t^D$, is passed through an affine transform followed by the softmax function, *SM*, to form $P(w_t\,|\,w_{t-1}...w_0, \mathbf{p})$. A word w_t can be obtained by either taking the word with the highest probability or sampling from the distribution:

$$P(w_t \mid w_{t-1}...w_0,\mathbf{p}) = SM(W_w\mathbf{h}_t^D + \mathbf{b}_w) \tag{4.4}$$

$$w_t \sim P(w_t \mid w_{t-1}...w_0, \mathbf{p}) \tag{4.5}$$

During training, the words are not sampled from the output distribution, but instead the true words from the dataset are used. This a common technique that is often referred to as *teacher-forcing*, though it also directly follows from equation 4.3.

To generate a sequence using an RNN, beam-search is often used. Using beam-search with n beams, the words corresponding to the top n probabilities of $P(w_0 \,|\, \mathbf{p})$ are the first n beams. For each succeeding w_t, the n words corresponding to the top n probabilities of $P(w_t \,|\, w_{t-1}...w_0, \mathbf{p})$ are taken for each of the n beams. This is followed by reducing the number of beams from now n^2 down to n, by taking the n beams with the highest probability $P(w_t w_{t-1}...w_0 \,|\, \mathbf{p})$. This is a deterministic process. However, it is not realistic for the NUS to always give the same response in the same context. Thus, the NUS cannot cover the full breadth of user behaviour if beam-search is used. To solve this issue while keeping the benefit of rejecting sequences with low probability, a type of beam-search with sampling is used. The process is identical to the above, but n words per beam are sampled from the probability distribution. The NUS is now non-deterministic resulting in a diverse US. Using two beams gave a good trade-off between reasonable responses and diversity.

4.4 Training

The neural sequence-to-sequence model is trained to maximize the log probability that it assigns to the user utterances of the training data set:

$$\mathcal{L} = \sum_{n=1}^{N} \log P(w_0 \,|\, \mathbf{p}) \sum_{t=1}^{L_n} \log P(w_t \,|\, w_{t-1:0}, \mathbf{p}) \tag{4.6}$$

The network was implemented in Tensorflow (Abadi et al., 2015) and optimized using Tensorflow's default setup of the Adam optimizer (Kingma and Ba, 2015). The LSTM layers and the fully-connected layer had widths of 100 each to give a reasonable number of overall parameters. The width was not tuned. The learning rate was optimised on a held out validation set and no regularization methods used. The training set was shuffled at the dialogue turn level.

Chapter 5

Experiments

5.1 DSTC2 Dataset

The Second Dialog State Tracking Challenge (DSTC2) (Henderson et al., 2014) was set up to provide a common testbed for dialogue state tracking, otherwise known as belief tracking. The domain of DSTC2 is a 'restaurant information' domain set in the city of Cambridge, UK. The dataset consists of dialogues between a user and a prototype of a Spoken Dialogue System. The dataset is extensively labelled. Each turn has both transcriptions and their corresponding semantics for the user utterance. Furthermore, each turn is labelled with the portion of the goal that the user has expressed up until and including the current turn. These turn-specific labels can be turned into goal labels whose behaviour is the same as it would be if a policy were trained (see Section 4.1.1).

When training both the sequence-to-sequence model of the NUS (Section 4.3) and the SC-LSTM (Section 3.3.2) the target utterances were the manual transcriptions and not the ASR output. Since the transcriptions were done manually, they contained spelling errors. These were manually corrected to ensure proper delexicalization. Some dialogues were discarded due to transcription errors being too large. After cleaning the dataset, the training set consisted of 1609 dialogues with a total of 11638 dialogue turns. The validation set had 505 dialogues with 3896 dialogue turns. The maximum sequence length of the delexicalized turns was 22, including the end of sentence character. The maximum dialogue length was 30 turns.

5.2 Experimental Setup

The evaluation of user simulators is an ongoing area of research and a variety of techniques can be found in the literature. Most papers published on user simulation evaluate their US using *direct* methods. These methods evaluate the US through a statistical measure of similarity between the outputs of the US and a real user on a test set. Multiple models can outperform the ABUS on these metrics. However, this is unsurprising since these user simulators were trained on the same or similar metrics. The ABUS was explicitly proposed as a tool to train the policy of a dialogue manager and it is still the dominant form of US used for this task. Therefore, the only fair comparison between a new US model and the ABUS is to use the *indirect*, utility driven method of evaluating the policies that were obtained by training with each US. The NUS is also compared to an ABUS augmented with a Natural Language Generator (NLG), described in Section 3.3.2, termed ABUS-NLG.

5.2.1 Training of Dialogue Policies

All dialogue policies were trained within the framework of the PyDial toolkit (Ultes et al., 2017), by interacting with either the NUS, the ABUS or ABUS-NLG. The RL algorithm used is GP-SARSA (Gašić and Young, 2014) with hyperparameters taken from (Casanueva et al., 2017). The reward function used gives a reward of 20 to a successfully completed dialogue and of -1 for each dialogue turn. The maximum dialogue length was 25 turns. The presented metrics are success rate (SR) and average reward over test dialogues. SR is the percentage of dialogues for which the system satisfied both the user's constraints and requests. The final goal, after possible goal changes, was used for this evaluation. When policies are trained using the NUS or the ABUS-NLG, its output is parsed using PyDial's regular expression based semantic decoder. The policies were trained for 4000 dialogues.

5.2.2 Testing with simulated users

In Schatztmann et al. (2005) *cross-model evaluation* is proposed to compare user simulators. First, the user simulators to be evaluated are used to train N policy each. Then these policies are tested using the different user simulators and the results averaged. Schatztmann et al. (2005) showed that a strategy learned with a good user simulator still performs well when tested on poor user simulators. At the same time policies learned with a poor user simulator still performed well on that user simulator. If a policy performs well on all user simulators and not just on the one that it was trained on, it indicates that the US with which it was trained is diverse and realistic, and thus the policy is likely to perform better on real users. For each US five policies ($N = 5$), each using a different random seed for initialisation, are trained. Results are reported for both the best and the average performance on 1000 test dialogues. The ABUS is programmed to always mention the new goal after a goal change. In order to not let this affect the performance of the policies trained with the ABUS and ABUS-NLG too negatively, the same is implemented for the NUS by re-sampling a sentence if the new goal is not mentioned.

5.2.3 Testing with real users

Though the above test is already more indicative of policy performance on real users than measuring statistical metrics of user behaviour, a better test is to test with human users. Even though real user studies are common in Dialogue research overall, for research on user simulation they are done surprisingly rarely. For the test on human users, only ABUS and NUS are compared. The human evaluation was part of a publication for SIGDIAL 2018 (Kreyssig et al., 2018a). Due to time and budget constraints human evaluation using the ABUS-NLG was not done. Two policies for each US that was used for training are chosen from the five policies. The first policy is the one that performed best when *tested on the NUS*. The second is the one that performed best when *tested on the ABUS*. This choice of policies is motivated by a type of overfitting to be seen in Sec. 5.3.1. The evaluation of the trained dialogue policies in interaction with real users follows a similar set-up to (Jurčíček et al., 2011). Users are recruited through the Amazon Mechanical Turk (AMT) service. 1000 dialogues (250 per policy) were gathered. The learnt policies were incorporated into an SDS pipeline with a commercial ASR system. The AMT users were asked to find a restaurant that matches certain constraints and find certain requests. Subjects were randomly allocated to one of the four analysed systems. After each dialogue, the users were asked whether they judged the dialogue to be successful or not which was then translated to the reward measure.

5.3 Experimental Results

5.3.1 Cross-Model Evaluation

US for Training	US for Evaluation					
	NUS		ABUS		ABUS-NLG	
	Rew.	SR	Rew.	SR	Rew.	SR
NUS-best	13.0	98.0$^{\mathcal{N}_1}$	13.3	99.8	12.6	98.1
ABUS-best	1.5	71.5$^{\mathcal{A}_1}$	13.8	99.9$^{\mathcal{A}_2}$	12.4	96.6
ABUS-NLG-best	8.1	84.1	13.7	99.6	13.5	98.1
NUS-avg	12.4	96.6	11.2	94.0	10.3	93.5
ABUS-avg	-7.6	45.5	13.5	99.5	12.3	95.8
ABUS-NLG-avg	-3.5	54.8	13.5	99.3	13.1	97.4

Table 5.1: Results for policies trained for 4000 dialogues on NUS, ABUS and ABUS-NLG when tested on these USs for 1000 dialogues. Five policies with different initialisations were trained for each US. Both average and best results are shown. $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{N}_1$ are used for the human evaluation.

Table 5.1 shows the results of the cross-model evaluation after 4000 training dialogues. The policies trained with the NUS achieved an average success rate (SR) of 96.6%, 94.0% and 93.5% when tested on the NUS, ABUS and the ABUS-NLG, respectively. By comparison, the policies trained with the ABUS achieved average SRs of 45.5%, 99.5% and 95.8%, respectively. The policies trained with the ABUS-NLG achieved average SRs of 54.8%, 99.3% and 97.4, respectively. Thus, training with the NUS leads to policies that can perform well on all user simulators, which is not the case for training with the ABUS. Furthermore, the best SRs when tested on the ABUS are similar for all USs at 99.8% (NUS), 99.9% (ABUS) and 99.6% (ABUS-NLG). When tested on the NUS the best SRs were 98.0% (NUS), 71.5% (NUS) and 84.1% (NUS). This shows that the behaviour of the Neural User Simulator is realistic and diverse enough to train policies that can also perform very well on the Agenda-Based User Simulator.

For NUS and ABUS, of the five policies, for each US, the policy performing best on the NUS was not the best performing policy on the ABUS. Similarly, of the five policies trained with the ABUS-NLG, the policy performing best on the NUS was not the policy performing best on the NUS. This could indicate that the policy "overfits" to a particular user simulator. Overfitting would manifest itself in worse results on a different US as the model is trained for longer.

Five policies trained on each US for only 1000 dialogues were also evaluated, the results of which can be seen in Table 5.2. After training for 1000 dialogues, the average SR of the policies trained on the NUS when tested on the ABUS was 97.3% in comparison to 94.0% after 4000 dialogues. Similarly, when tested on the ABUS-NLG the average SR is 95.1% compared to 93.5% after 4000 dialogues. This indicates that the policies did indeed overfit to the NUS. For the ABUS-NLG, this was observed when tested on the NUS with an average SR of 62.4% compared to 54.8% after 4000 dialogues. When tested on the ABUS, the ABUS-NLG improved its performance when training for the additional 3000 dialogues, which is expected due to the similarity in user behaviour. For the policies trained with the ABUS, this was not observed.

US for Training	US for Evaluation					
	NUS		ABUS		ABUS-NLG	
	Rew.	SR	Rew.	SR	Rew.	SR
NUS-best	12.2	95.9	13.9	99.9$^{\mathcal{N}_2}$	12.4	98.0
ABUS-best	-4.0	54.8	13.2	99.0	91.0	10.85
ABUS-NLG-best	6.6	86.5	12.3	99.2	13.2	97.8
NUS-avg	12.0	95.4	12.2	97.3	10.6	95.1
ABUS-avg	-9.5	42.3	12.8	98.4	10.3	88.8
ABUS-NLG-avg	-3.0	62.4	11.9	98.6	12.1	94.3

Table 5.2: As Table 5.1 but the policies are only trained for 1000 dialogues. $\mathcal{N}_2$ is used for human evaluation.

5.3.2 Human Evaluation

The results of the human evaluation are shown in Table 5.3 for 250 dialogues per policy. In Table 5.3 policies are marked using an ID ($\mathcal{U}_\alpha$) that translates to results in Tables 5.1 and 5.2. Both policies trained with the NUS outperformed those trained on the ABUS in terms of both reward and success rate. The best performing policy trained on the NUS achieves a 93.4% success rate and 13.8 average rewards whilst the best performing policy trained with the ABUS achieves only a 90.0% success rate and 13.3 average reward. This shows that the good performance of the NUS on the cross-model evaluation transfers to real users. Furthermore, the overfitting to a particular US is also observed in the real user evaluation. For not only the policies trained on the NUS, but also those trained on the ABUS, the best performing policy was the policy that performed best on the other US. The policy both trained and optimized (in terms of the random seed) on the ABUS, which is common practice found in many publications, only achieved a success rate 88.5%.

Training Simulator	Human Evaluation	
	Reward	Success Rate
NUS - $\mathcal{N}_1$	13.4	91.8
NUS - $\mathcal{N}_2$	13.8	93.4
ABUS - $\mathcal{A}_1$	13.3	90.0
ABUS - $\mathcal{A}_2$	13.1	88.5

Table 5.3: Real User Evaluation. Results over 250 dialogues with human users. $\mathcal{N}_1$ and $\mathcal{A}_1$ performed best on the NUS. $\mathcal{N}_2$ and $\mathcal{A}_2$ performed best on the ABUS. Rewards are not comparable to Table 5.1 and 5.2 since all user goals were achievable.

Chapter 6

Conclusions

6.1 Summary

In order to build and train large Spoken Dialogue System, methods are needed to train these systems based on a corpus rather than in interaction with real users. The principal method used for corpus-based learning of dialogue strategies is to use user simulators, which motivates their research. The limitations, discussed in Sections 1.2, 2.2, and 2.3, of previously proposed user simulators originate from modelling user behaviour on the semantic rather than the word level, insufficient modelling of the dialogue history, rule-based and hand-picked rather than learned user behaviour, lacking an explicit goal representation or requiring too much engineering effort and domain specific knowledge.

In this project, the Neural User Simulator (NUS) was developed and tested. The NUS uses the system's response in its semantic form as input and gives a response in the form of natural language. Due to its Deep Learning based sequence-to-sequence architecture it can model the dialogue history well by learning what constitutes a good user state rather than using a predefined user state. The behaviour of the NUS is entirely learned from a corpus of recorded dialogues and it being explicitly conditioned on a goal allows for computing the reward measure mandatory for policy optimisation. Furthermore, the NUS learned the challenging task of reacting to user goal changes during a dialogue, a task that has been largely ignored by previous works. Furthermore, the NUS can be applied to any dialogue domain with minimal engineering effort as long as sufficient in-domain data is available.

It was shown that the NUS learns realistic user behaviour from a corpus of recorded dialogues such that it can be used to optimise the policy of the dialogue manager of a spoken dialogue system. The NUS was compared to two user simulators, first the Agenda-Based User Simulator and secondly an Agenda-Based User Simulator augmented with a Natural Language Generator. This comparison was done by evaluating policies trained with these user simulators. The trained policies were compared both by testing them with simulated users and also with real users. The NUS excelled on both evaluation tasks. This was the first time that the Agenda-Based User Simulator, the de facto state of the art in user simulation, was outperformed when comparing the policies trained with it and another user simulator. Beyond this success, there are more advantages to the NUS. First, due to generating natural language it can be used to train an SDS that uses a text-based Belief Tracker. Due to text-based Belief Tracking being a prominent area of research, this is a significant contribution. Secondly, the word-level output of the NUS allows for text-based error modelling. Due to the nature of ASR errors, this should be more suitable than semantics-based error modelling. Thirdly, when collecting a corpus, only transcriptions of the user response are needed and not their corresponding semantics. This reduces the cost per dialogue when collecting a corpus, allowing for larger corpora to be collected.

6.2 Limitations

The advantage of the NUS is that it can be built with very limited domain specific knowledge by learning the entire behaviour directly from data. However, this also means that its behaviour is confined to the data it was trained with. Even though the NUS was able to generalize to unseen dialogues, those were still similar to those within the used corpus. It is unlikely that if the prototype used to collect the corpus were already nearly optimal that the NUS would have been able to cope with a DM that is trained from scratch since the corpus would not contain data of interactions with the DM that gives bad, seemingly random responses.

Furthermore, the parameters (i.e. weights) of the NUS are non-interpretable. It is essentially impossible to determine the meaning of each of the parameters. Thus, it is not possible to tune its behaviour. The Agenda-Based User Simulator by comparison has highly interpretable parameters, which in most cases are set by hand. For example the Agenda-Based User Simulator can be made slightly more "Unfriendly", meaning that it does not give out much information if not directly prompted. This easy change of User Simulator helps with the research regarding reinforcement learning algorithms for policy optimisation. Policies can be trained with one "version" of the Agenda-Based User Simulator and tested on another to evaluate how much the RL algorithm can transfer to these unseen environments as done by Casanueva et al. (2017, 2018). This is not possible with the NUS in its current form.

6.3 Future Work

Even though the NUS reduced the necessary annotation effort for corpora collection by not requiring semantically annotated user responses, transcriptions are still needed. A superior approach would be to directly predict the noisy ASR output and an associated confidence score. This would eliminate all necessary annotation. The only necessary labels are the user goal and possible goal changes, which can be generated automatically without human interference. By far larger corpora can be collected since the only cost would be to pay users to talk to a prototype system.

In this project the experiments evaluated the NUS in the Cambridge restaurant domain. This domain is relatively small, having only three informable slots. Future work should focus on applying the NUS to larger domains or even better multi-domain tasks. This is likely to involve some change of model architecture so that some model parameters are shared and others are not shared between the different domains.

The NUS is a model that tries to generate data that is similar to the dataset used to train it. However, this project trained the NUS in a discriminative fashion by using a standard cross-entropy loss as done for classification tasks. For other complex tasks that try to generate data similar to the training data such as image generation, Variational Auto-Encoder's (VAE) (Kingma and Welling, 2014) or Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) are often used. Recently, Tseng et al. (2018) showed that using the framework of a conditional-VAE can improve the SC-LSTM model (see Section 3.3.2) for Natural Language Generation. Thus, a natural extension of this project would be to try an approach using a GAN or a VAE.

Lastly, it would also be important to use the NUS to train a reinforcement learning based SDS that uses a text-based belief tracker in a simulated environment or use it as a model in model-based reinforcement learning.

Bibliography

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

Masahiro Araki, Taro Watanabe, and Shuji Doshita. 1997. Evaluating dialogue strategies for recovering from misunderstandings. In *Proc. IJCAI Workshop on Collaboration, Cooperation and Conflict in Dialogue Systems*.

Lalit Bahl, Peter Brown, Peter De Souza, and Robert Mercer. 1986. Maximum mutual information estimation of hidden markov model parameters for speech recognition. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'86.*, volume 11, pages 49–52. IEEE.

Harry C. Bunt. 1981. Rules for the interpretation, evaluation and generation of dialogue acts. *IPA annual progress report 16*, pages 99–107.

Iñigo Casanueva, Paweł Budzianowski, Florian Kreyssig, Stefan Ultes, Bo-Hsian Tseng, Yen-Chen We, and Milica Gašić. 2018. Feudal dialogue management with jointly learned feature extractors. In *Proceedings of the 19th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.

Iñigo Casanueva, Paweł Budzianowski, Pei-Hao Su, Nikola Mrkšić, Tsung-Hsien Wen, Stefan Ultes, Lina Rojas-Barahona, Steve Young, and Milica Gašić. 2017. A benchmarking environment for reinforcement learning based task oriented dialogue management. In *NIPS Deep Reinforcement Learning Symposium*.

Senthilkumar Chandramohan, Matthieu Geist, Fabrice Lefevre, and Olivier Pietquin. 2011. User simulation in dialogue systems using inverse reinforcement learning. In

Proceedings of the Twelfth Annual Conference of the International Speech Communication Association.

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

Paul Crook and Alex Marin. 2017. Sequence to sequence modeling for user simulation in dialog systems. In *Proceedings of the 18th Annual Conference of the International Speech Communication Association.*

Heriberto Cuayáhuitl, Steve Renals, Oliver Lemon, and Hiroshi Shimodaira. 2005. Human-computer dialogue simulation using hidden markov models. In *Automatic Speech Recognition and Understanding, 2005 IEEE Workshop on*, pages 290–295. IEEE.

Wieland Eckert, Esther Levin, and Roberto Pieraccini. 1997. User modeling for spoken dialogue system evaluation. In *Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on*, pages 80–87. IEEE.

Layla El Asri, Jing He, and Kaheer Suleman. 2016. A sequence-to-sequence model for user simulation in spoken dialogue systems. *Proceedings of the 17th Annual Conference of the International Speech Communication Association*, pages 1151–1155.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. 2010. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660.

Milica Gašić and Steve Young. 2014. Gaussian processes for pomdp-based dialogue manager optimization. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(1):28–40.

M. Gašić, F. Jurčíček, B. Thomson, K. Yu, and S. Young. 2011. On-line policy optimisation of spoken dialogue systems via live interaction with human subjects. In *Automatic Speech Recognition and Understanding, 2011 IEEE Workshop on.*

Kallirroi Georgila, James Henderson, and Oliver Lemon. 2005. Learning user simulations for information state update dialogue systems. In *Ninth European Conference on Speech Communication and Technology.*

Sharon Goldwater, Dan Jurafsky, and Christopher D Manning. 2010. Which words are hard to recognize? prosodic, lexical, and disfluency factors that increase speech recognition error rates. *Speech Communication*, 52(3):181–200.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*.

Matthew Henderson, Blaise Thomson, and Jason D Williams. 2014. The second dialog state tracking challenge. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pages 263–272.

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

Filip Jurčíček, Simon Keizer, Milica Gašić, Francois Mairesse, Blaise Thomson, Kai Yu, and Steve Young. 2011. Real user evaluation of spoken dialogue systems using amazon mechanical turk. In *Proceedings of the Twelfth Annual Conference of the International Speech Communication Association*.

Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. 2016. Grid long short-term memory. In *Proceedings of the International Conference on Learning Representations*, San Juan.

Simon Keizer, Milica Gašić, Filip Jurčíček, François Mairesse, Blaise Thomson, Kai Yu, and Steve Young. 2010. Parameter estimation for agenda-based user simulation. In *Proceedings of the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 116–123. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of The International Conference on Learning Representations (ICLR)*.

Diederik P Kingma and Max Welling. 2014. Auto-encoding variational bayes. In *Proceedings of The International Conference on Learning Representations*.

Florian Kreyssig, Iñigo Casanueva, Paweł Budzianowski, and Milica Gašić. 2018a. Neural user simulation for corpus-based policy optimisation of spoken dialogue systems. In *Proceedings of the 19th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.

Florian Kreyssig, Chao Zhang, and Philip C. Woodland. 2018b. Improved TDNNs using deep kernels and frequency dependent grid-rnns. In *Proceedings of ICASSP*.

Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz. 2017. End-to-end task-completion neural dialogue systems. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 733–743, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Bing Liu and Ian Lane. 2017. Iterative policy learning in end-to-end trainable task-oriented neural dialog models. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop, ASRU*, pages 482–489.

Nikola Mrkšić, Diarmuid Ó Séaghdha, Tsung-Hsien Wen, Blaise Thomson, and Steve Young. 2017. Neural belief tracker: Data-driven dialogue state tracking. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1777–1788.

Olivier Pietquin and Thierry Dutoit. 2006. A probabilistic framework for dialog simulation and optimal strategy learning. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(2):589–599.

Daniel Povey and Philip C Woodland. 2002. Minimum phone error and i-smoothing for improved discriminative training. In *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, volume 1, pages I–105. IEEE.

Osman Ramadan, Paweł Budzianowski, and Milica Gašić. 2018. Large-scale multi-domain belief tracking with knowledge sharing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics.

Nicholas Roy, Joelle Pineau, and Sebastian Thrun. 2000. Spoken dialogue management using probabilistic reasoning. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 93–100. Association for Computational Linguistics.

Jost Schatzmann. 2008. *Statistical User and Error Modelling For Spoken Dialogue Systems.* Ph.D. thesis, University of Cambridge.

Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. 2007. Agenda-based user simulation for bootstrapping a pomdp dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 149–152. Association for Computational Linguistics.

Jost Schatzmann, Karl Weilhammer, Matt Stuttle, and Steve Young. 2006. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *The knowledge engineering review*, 21(2):97–126.

Jost Schatztmann, Matthew N Stuttle, Karl Weilhammer, and Steve Young. 2005. Effects of the user model on simulation-based learning of dialogue strategies. In *Automatic Speech Recognition and Understanding, 2005 IEEE Workshop on*, pages 220–225. IEEE.

Konrad Scheffler and Steve Young. 2000. Probabilistic simulation of human-machine dialogues. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, volume 2, pages II1217–II1220. IEEE.

Konrad Scheffler and Steve Young. 2001. Corpus-based dialogue simulation for automatic strategy learning and evaluation. In *Proc. NAACL Workshop on Adaptation in Dialogue Systems*, pages 64–70.

John R Searle. 1969. *Speech acts: An essay in the philosophy of language.* Cambridge university press.

Manex Serras, María Inés Torres Torres, and Arantza del Pozo. 2017. Regularized neural user model for goal oriented spoken dialogue systems. In *International Workshop on Spoken Dialogue Systems.* Association for Computational Linguistics.

Kai Sun, Lu Chen, Su Zhu, and Kai Yu. 2014. The sjtu system for dialog state tracking challenge 2. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pages 318–326.

Kai Sun, Qizhe Xie, and Kai Yu. 2016. Recurrent polynomial network for dialogue state tracking. *Dialogue & Discourse*, 7(3):65–88.

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

David R Traum. 1999. Speech acts for dialogue agents. In *Foundations of rational agency*, pages 169–201. Springer.

Bo-Hsian Tseng, Florian Kreyssig, Paweł Budzianowski, Iñigo Casanueva, Yen-Chen Wu, Stefan Ultes, and Milica Gašić. 2018. Variational cross-domain natural language generation for spoken dialogue systems. In *Proceedings of the 19th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL).*

Stefan Ultes, Lina M. Rojas Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Iñigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, Milica Gašić, and Steve Young. 2017. PyDial: A Multi-domain Statistical Dialogue System Toolkit. In *Proceedings of ACL 2017, System Demonstrations*, pages 73–78, Vancouver, Canada. Association for Computational Linguistics.

Taro Watanabe, Masahiro Araki, and Shuji Doshita. 1998. Evaluating dialogue strategies under communication errors using computer-to-computer simulation. *IEICE transactions on information and systems*, 81(9):1025–1033.

Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1711–1721.

Jason D Williams. 2008. Evaluating user simulations with the cramér–von mises divergence. *Speech communication*, 50(10):829–846.

Jason D Williams and Steve Young. 2007. Partially observable markov decision processes for spoken dialog systems. *Computer Speech & Language*, 21(2):393–422.

Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. 2013. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179.

Lukas Zilka and Filip Jurcicek. 2015. Incremental lstm-based dialog state tracker. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 757–762. IEEE.