Institut für Maschinelle Sprachverarbeitung
Universität Stuttgart
Pfaffenwaldring 5B
D-70569 Stuttgart

Neural-based Methods for User Simulation in Dialog Systems

Maximilian Schmidt

Master thesis

Studiengang:	Informatik M.Sc.
Prüfer:	Prof. Dr. Ngoc-Thang Vu
	Dr. Antje Schweitzer
Betreuer:	Prof. Dr. Ngoc-Thang Vu
Beginn der Arbeit:	01.06.2018
Ende der Arbeit:	30.11.2018

Abstract

Spoken Dialog Systems allow users to interact with a Dialog Manager (DM) using natural language, thereby following a goal to fulfill their task. State-of-the-art solutions cast the problem as Markov Decision Process, leveraging Reinforcement Learning (RL) algorithms to find an optimal dialog strategy for the DM. For this purpose, several thousand dialogs need to be seen by the RL agent. A user simulator comes in handy to generate responses on demand, however the current state-of-the-art agenda-based user simulators lack the ability to model real human subjects. In this thesis, this problem is addressed by implementing a user simulator using a Recurrent Neural Network which approximates the agenda-based model in a first step. Going onwards, it is shown to learn noise and variance treated as varying user behavior. This is used to train the simulator on real data thus modeling real users.

Zusammenfassung

Spoken Dialog Systems ermöglichen es Nutzern mittels Sprache oder Text, Aufgaben zu erledigen oder einfachen Zugang zu einer Datenbank zu erhalten. State-of-the-art Ansätze modellieren dieses Problem als Markov Decision Process. Dies ermöglicht den Einsatz von Reinforcement Learning (RL) Algorithmen, um eine optimale Strategie für den Dialog Manager zu finden. Dafür muss der RL Agent allerdings etliche Dialoge sichten. Ein Nutzersimulator generiert zu diesem Zweck die benötigten Antworten. Der state-of-the-art, Agenda basierte Nutzersimulator kann Nutzer jedoch nicht realitätsnah nachbilden. Diese Masterarbeit versucht das Problem zu lösen, indem ein Nutzersimulator mittels eines Recurrent Neural Networks implementiert wird. Es wird zuerst gezeigt, dass dieser den vorhandenen, händischen Nutzersimulator nachbildet. Weiterhin wird gezeigt, dass dieses Modell Rauschen (dargestellt als individuelles Nutzerverhalten) lernen und damit auch auf echten Daten trainiert werden kann, um Nutzer menschlich modellieren zu können.

Contents

List of Figures

List of Tables

Abbreviations

A2C Advantage Actor Critic

ASR Automatic Speech Recognition

DM Dialog Manager

DQN Deep Q Network

DSTC2 Dialog State Tracking Challenge 2

eNAC episodic Natural Actor Critic

IRL Inverse Reinforcement Learning

LSTM Long Short-Term Memory

MDP Markov Decision Process

NLG Natural Language Generation

NLU Natural Language Understanding

NN Neural Network

ReLU Rectified Linear Unit

RL Reinforcement Learning

RNN Recurrent Neural Network

SDS Spoken Dialog System

SGD Stochastic Gradient Descent

TTS Text-To-Speech

WOZ Wizard of Oz

1 Introduction

To give an overview about the problem setting and what this thesis aims for, this section will introduce Spoken Dialog Systems (SDSs) on a high level.

1.1 Dialog Systems

Due to recent advances in machine learning, SDSs have emerged in a variety of domains. SDSs allow users to interact with a computer using text or speech input in order to get access to a database. The dialog system should recognize the user's intention and answer appropriately. Although in current commercial (proprietary) applications the system tries to answer a user demand, e.g. telling a joke or giving predefined answers similar to FAQs, they usually do not cover whole conversations. In research, on the other hand, they often appear as task-oriented systems where full dialogs form a conversation. In task-oriented SDSs, the user follows a goal (e.g. finding the address of a french restaurant in the south of town) which the system should fulfill. For this purpose, the user tells the system about so-called constraints (e.g. food=french, area=south) and requests (e.g. address). The system, on the other hand, is concerned with determining the information the user wants in order to provide it to the user (Bunt, 1981b). Those constraints and requests are reflected in the ontology of a dialog system.

The ontology specifies a domain and makes all relevant information concerning this domain available to the dialog system. It therefore holds a database with possible entities within the domain and their properties. The ontology for the PyDial[1] framework used in this thesis as well as the framework itself is shown in more detail in section 4.1.

Within the course of the dialog, the system recognizes these inform actions and requests more information from the user if needed to find an appropriate answer to the users goal (which the system doesn't know). However, the

[1] http://www.camdial.org/pydial/

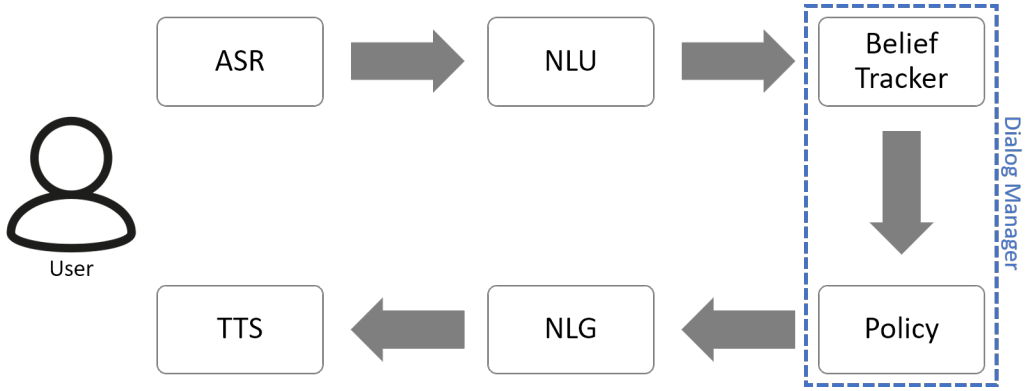

Figure 1: Process within a Spoken Dialog System (SDS): A user's utterance is converted through ASR and NLU to a dialog act, which is the input for the Dialog Manager (DM). Last updates its belief based on this input and decides on the next action (policy), which traverses the NLG and TTS until the user gets the answer spoken. (adapted from Williams et al. (2016))

actions within a dialog are not limited to *inform* and *request* but also include discourse actions such as *hello* or *repeat*.

An SDS usually consists of several modules, namely Automatic Speech Recognition (ASR), Natural Language Understanding (NLU), Dialog Manager (DM) (belief tracker & policy), Natural Language Generation (NLG) and Text-To-Speech (TTS). In case of speech input to the SDS, the ASR processes the input. Once there is a textual representation of the input, the NLU tries to extract a semantic representation for the DM. The semantic representation covers only the raw facts. The DM in turn holds and updates a belief state, modeling also uncertainty induced in the prior modules by noise. Using the state, an action according to a policy is selected and issued to the user while converted into text (NLG) and speech (TTS component) if applicable. The whole interaction is depicted in fig. 1.

1.2 Motivation

This thesis aims to provide contributions to methods which model the problem as a Markov Decision Process (MDP). This enables the DM to find an optimal policy (w.r.t. a specific criterion) using strategies deployed in MDP. Most important to mention are Reinforcement Learning (RL) algorithms, some of them which make use of deep learning. The idea of RL is to find such an optimal policy using an iterative *trial and error* process. For this purpose, the agent chooses an action according to a policy (sometimes randomly to explore the state and action space), which is used by the environment (a user (simulator) in case of dialog systems) to bring the agent in a new state while giving feedback (reward) on the taken action. The reward tells the agent about the quality of this action. The higher the reward, the better the action in the corresponding state. This is visualized in fig. 2.

As this training is an iterative process, the agent needs a lot of training data. In an SDS, this means to get an appropriate answer from the user whenever the system makes a trial in the form of issuing an action. Since there is typically the need for thousands of dialogs to be seen by the RL agent, it is intractable to let human subjects decide on the answer and give the necessary input to the agent. The idea of a user simulator is to generate those dialogs on demand.

1.3 Contribution

The principal contribution of this thesis is to provide a neural-based user simulator trained using supervised learning. The idea of neural-based user simulators is to use a small amount of data available to train a Neural Network (NN). Later on, the user simulator can yet be used with different dialogs the simulator has never seen. This is made possible due to the ability of NNs to generalize beyond seen data. Additionally, a trainable user simulator is important in SDSs to train the whole system end-to-end instead of each

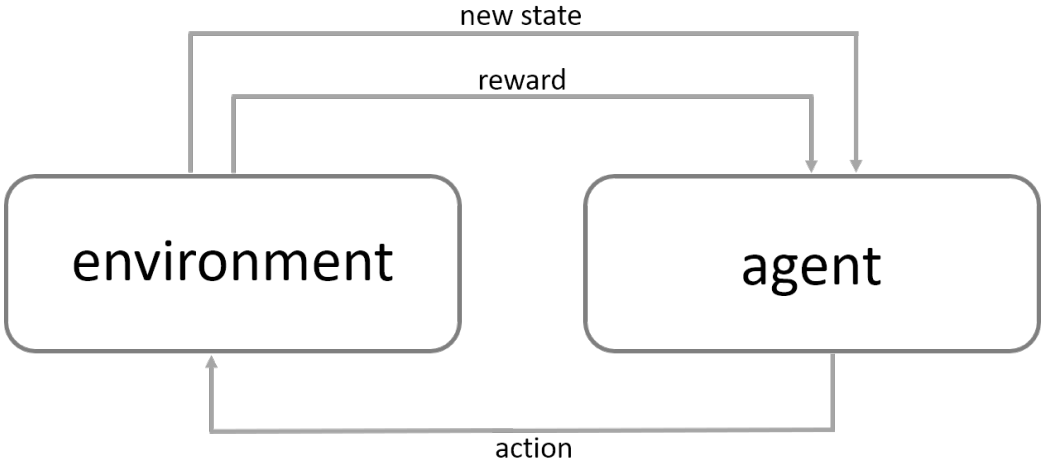

Figure 2: The Reinforcement Learning (RL) approach: an agent is currently in a state, samples an action and gets feedback from the environment in the form of a new state and a reward. The reward is a measure of quality for the taken action. (adapted from Sutton and Barto (1998))

module individually. This incorporates to jointly train a user simulator with the dialog manager as shown in Liu and Lane (2017a). To allow benchmarks with other (often handcrafted) simulators and to be able to evaluate a learned policy trained with this neural-based user simulator, it is integrated into the PyDial framework (Ultes et al., 2017). The user simulator is trained from generated, synthetic data as well as from data collected using real human subjects and shown to learn a reasonable distribution over the user actions. This is very handy when it comes down to deploying the SDS in a real environment, where individual behavior is inherently hidden in the training data from real users which should be used to model human subjects more accurately. Experiments are conducted to evaluate its performance and compare it with the handcrafted user simulator, from which the data has been generated.

Additionally, another environment to the PyDial framework is proposed to boost the performance of policies trained on this environment as it gives

more control about the noise. It can be easily shown that a policy with high success rate and reward trained with a handcrafted user simulator (where no real data was used) fails in the real world.

This behavior occurs due to the fact that it's easy to evaluate a trained policy in terms of a dialog's success, but it has to be in mind that those metrics depend on the user. However, it is difficult to say whether a real user will behave similar to the simulated user and whether both conform in terms of the metrics since the state space needed for real users is probably much larger.

To complete the work of this thesis, a user simulator trained on real data is provided too. This will prove useful for the aforementioned problem.

The thesis is structured as follows:

Related work concerning user simulators and the theory about deep learning is covered in sections 2 and 3. Section 4 introduces the resources used in this work, whereas section 5 describes the neural-based model employed for user simulation. Section 6 demonstrates the user simulator and highlights the findings. A conclusion in section 7 and an outlook in section 8 complete this thesis.

2 User Simulation for Spoken Dialog Systems

Within this section, SDSs are formally defined and an overview to existing user simulation methods is given.

2.1 Dialog Systems

In current research, the policy of an SDS is trained using data-driven methods, primarily RL. For this purpose, the dialog setting is cast as an MDP (Levin et al., 1997; Daubigney et al., 2012).

An MDP is a tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ is the set of all possible states, $\mathcal{A}$ the set of all possible actions, T the stochastic transition function yielding the probability for the next state s' defined as $T(s, a, s') = P(s' \mid s, a) \in [0, 1]$, R the real-valued reward given by $R(s, a, s')$ and a discount factor $\gamma \in [0, 1)$. The latter is used to formally bound the number of steps an agent can take within one episode. It could also be seen as preferring immediate rewards over rewards in the future course of the sequence or modeling the agents lifetime (Kaelbling et al., 1996). An episode is a sequence of states $s_0, s_1, \ldots, s_T$ where s_{t+1} is given by $T(s_t, a, s_{t+1})$ satisfying the Markov property

$$P(s_{t+1} \mid s_t, s_{t-1}, \ldots, s_0) = P(s_{t+1} \mid s_t). \tag{1}$$

This implies that any state only depends on the last state since the history isn't needed.

Levin et al. (1997) describe the state space $\mathcal{S}$ as all possible values for the given slots of a domain. $\mathcal{A}$ contains actions such as opening and ending a dialog, asking the user for values or outputting the information requested by the user. The model T is not known in advance and not needed for RL as it belongs to the model-free methods. $R(s, a)$ is chosen such that the dialog policy acts in an efficient manner. The purpose of the discount factor is always the same through all MDPs thus there is no special definition needed. (Levin et al., 1997)

Since there is uncertainty in the input (e.g. through the ASR), the agent's state is seen as a belief state.

However, to learn a good policy, the RL agent needs thousands of dialogs to be seen because the state space is huge (Pietquin et al., 2011; Gašić et al., 2012). It is obvious that this problem is intractable if one wants to train and evaluate an agent with human subjects.

The need for user simulators is therefore inevitable and has several advantages. Besides the automatic training and evaluation on any number of dialogs, the user simulator enables to compare the evaluation of different dialog managers. Since there is less manual work involved in training the agent, the dialogs are also more consistent and less prone to errors. Furthermore, a user simulator's behavior can be guided to model the characteristics of different groups. (Eckert et al., 1997) This could prove reasonable if the dialog system will only target a specific audience.

2.2 Intention-level vs. text-level

Searle (1969) analyzed very early the acts of speech as modelling user intentions and Bunt (1981a) as modeling a dialog as a sequence of dialog acts. In general, it has to be differentiated between actions on the intention-level and on the text-level. In an example, a user issuing an action on the text-level would tell the system "I want to eat french food." This is the level on which human conversations usually operate (besides speech, but in an SDS speech is transformed into text). In contrast, the intention-level action carries only the actual information relevant to the system (Eckert et al., 1997). Following the example, an intention-level speech act[2] would be "inform(food=french)".

Eckert et al. (1997) motivate in their work the implementation of user simulators on the intention level. It represents the actual, solely important information. The speech and utterance level can thus be considered as different transport mechanisms. (Eckert et al., 1997)

[2] *action* and *act* are used interchangeably in this thesis

Additionally, it is always possible to add an NLG module on top to convert an intention-level action to a text-level action. This would also cover the variance inherent to human utterances. For example, Liu and Lane (2017b) use a template based NLG and Li et al. (2017) even generate those templates from data using a Long Short-Term Memory (LSTM). The template is then filled with the appropriate slots and values. This is often referred to as lexicalization. Wen et al. (2015) ameliorate this lexicalization task using an extended version of the LSTM to combine sentence planning (which outputs a template) and surface realisation (lexicalisation to generate real, natural text).

2.3 Different approaches to user simulation

There are several approaches regarding user simulation for SDSs. Before these are described, the requirements of user simulators as needed for this purpose are mentioned in this section.

2.3.1 Requirements

Pietquin and Hastie (2013) summarize the properties a user simulator should exhibit as follows: A user simulator should not only work well on available data, but also generalize beyond the seen data. This is important since the user simulator is seen as a bootstrapping method to generate more dialogs for training the dialog manager, as there is only a limited amount of training data available. Of course, there should be also some consistency on the sequence of actions or turns within a dialog as well as statistical consistency of the behavior of the simulated user with the data. That means it should produce an action (depending on e.g. the history of the dialog) with the same probability as it occurs in the training data. Less obvious is the property that the dialog system should perform overall well on a high level using the actions generated by the user simulator. Senseless repetitions or similar

actions could still end up successful, but won't produce an efficient policy on DM side. (Pietquin and Hastie, 2013)

2.3.2 N-grams

Different kind of models exist to implement a user simulator. N-grams are the most basic approach which model dialogs as a sequence of actions. The probability of taking an action is conditioned on the last $n-1$ actions (or information states) in a dialog. (Schatzmann et al., 2006) The parameters have to be derived from data, or manually forged. Although they could model the full history of a dialog, this would imply n to be large. However for large n not all possibilities will occur in the training data. In this case, one has to reduce n. (Georgila et al., 2006) This would end in a non-consistent behavior of the simulated user and there isn't any chance at all to follow a goal. For example, the rather common bi-gram model using only the last system action to condition the simulator's action may even produce dialog acts violating logical constraints (Schatzmann et al., 2006):

SYSTEM: What kind of food would you like?

USER: French of course!

SYSTEM: Sorry, I didn't understand. What did you say?

USER: Cheap please.

The context information is totally lost and the user cannot correctly answer the system's *repeat* act.

2.3.3 Graph-based models

Graph-based models, on the other hand, map all possible dialogs as paths to a network. They contain probabilistic choice nodes which model the stochasticity of an action. Those have to be set manually. (Schatzmann et al., 2006)

One of the first graph-based models is proposed by Scheffler and Young (2002) where they model the probability of taking a user action depending on an internal state. The representation of the internal state depends on a given user goal, thus leading to consistent dialogs. Their internal state representation looks as follows:

- The current constraints as given by the goal,
- the state of the requests in the goal (i.e. fulfilled or not fulfilled),
- how the user perceives the system's understanding of the user's goal and
- the last system prompt.

This gives a (manually crafted) feature vector depending on the current goal of the user. The probabilistic actions (i.e. the choice nodes) are then conditioned on this state. For this purpose, the probabilities as well as the choice nodes have to be set manually, resulting in excessive effort. Additionally, it is intractable to output any possible amount of actions, because this implies a huge amount of existing data. (Schatzmann et al., 2006) However, their approach is a first step to model consistent dialogs by explicitly modeling the user goal.

Schatzmann et al. (2007a) and Schatzmann and Young (2009) go one step further and build an agenda-based model. The idea is to initially build an agenda A depending on the user goal G, which covers all inform and request actions needed to fulfill this goal. For this purpose, the agenda is modelled as a stack where actions are pushed to and popped from the top. As the dialog evolves, the agenda may be augmented with more actions as the user interacts further with the system. When it is the user's turn to output an action, given probability $P(n \mid A, G)$ they pop n actions off the agenda A (which expresses goal G in terms of actions). The updates for A and G are modelled similarly. (Schatzmann et al., 2007a) The manual efforts required

for this approach do not only include maintaining the agenda (e.g. remove senseless actions or react on the last system prompt) but do also take into account the parameters, which have to be handcrafted. Additionally it lacks dynamic output for the user simulator because once an n is drawn from the probability distribution $P(n \mid A, G)$, n (fixed) actions are popped from the agenda regardless of the actions themselves. Obviously, the model supports multiple actions in the output, but those have no statistical consistency at all due to the aforementioned determination of actions and since n is often independent of A for tractability reasons. The authors also state that any probability distribution with its mode in the lower integer range (2-4) is sufficient for $P(n \mid A, G)$. In their extension in Schatzmann et al. (2007b) they learn the parameters for the stochastic model from data, instead of crafting them carefully by hand. Keizer et al. (2010) show in a similar approach how decision points - similar to other graph-based models - can be introduced, for which the probabilities can be learnt from data.

2.3.4 Machine-learning techniques

Last but by no means least, there are machine learning techniques which use regression to approximate the probability of choosing an action. This is an obvious remedy to the sparsity of available training data, often in the form of NNs. Those are universal approximators with the ability to generalize (Hornik et al., 1989; Kůrková, 1992).

Georgila et al. (2005) learn the weights for a linear function with a softmax applied to a state vector covering the state and the history of the dialog. More precisely, the state vector is a binary representation of 290 manually chosen features, like the presence of an information (e.g. user told about the food type they want) thus modeling the complete dialog history. (Schatzmann et al., 2006) The learning results in a probability distribution over the actions in each state. This model is similar to a one-layered NN with a softmax for the output.

El Asri et al. (2016) take one step further and learn a more complex Recurrent Neural Network (RNN) model on a series of context vectors, covering the history of the dialog. This deep learning approach has several advantages: it learns a representation from the training data, thus removing the need to carefully design the feature vector like Georgila et al. (2006). This implies a more robust model since human modeling is always prone to errors. On the other hand, their approach allows yet to follow a goal thus generating consistent dialogs by incorporating all information concerning the progress in terms of the goal into the feature vector, also known as context vector. Additionally, since the user action a_u is conditioned on the full history of the dialog, the probability distribution $P(a_u \mid c_t, \ldots, c_0)$ approximates the distribution in the original data. This comes in handy when sampling an action according to this distribution, resulting in higher variance in terms of the user simulator output.

Although the RL approach is indispensable in today's research concerning the dialog manager, there is little focus on applying those methods to user simulation. Chandramohan et al. (2011) proposed to use Inverse Reinforcement Learning (IRL) where the user action sequence is also cast as an MDP but the reward function is unknown. Therefore the goal is to learn some reward function from an expert (training data in this case) such that the learned policy exhibits a close behavior to the expert. This isn't an easy task and shifts the problem to finding a good reward function. Some relaxations like reducing the amount of possible actions to keep the state space small may help, but current research and accompanied experiments do not yield as high performance as other machine learning based user simulators. Therefore, the benefits of an IRL trained user simulator remain questionable. (Chandramohan et al., 2011)

3 Deep Learning

Since the proposed model relies on deep learning, mathematical fundamentals in this area are introduced and the deployed architectures are described.

3.1 Feed-forward networks

This section introduces NNs as the quintessential deep learning models and gives a short introduction into their functionality as well as an outline to existing models especially used in this thesis. The following theory is mainly based on a recent book by Goodfellow et al. (2016).

In classic machine learning based regression, the task is to learn a correlation between the input and output as seen in the training data. During training, the parameters ω of a linear model $\mathbf{y}_i = f(\mathbf{x}_i, \omega) = \mathbf{x}_i^T \omega \quad \forall i \in \{1, \ldots, n\}$ with $f : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$ are sought using a first or second order gradient optimization method, where $\mathbf{x}_i$ are the measured, independent variables and $\mathbf{y}_i$ the dependent variables, also known as labels. Using the kernel trick, where the input $\mathbf{x}_i$ is transformed into some higher dimensional space, it is possible to model non-linear correlations. For this purpose, a feature mapping ϕ has to be manually chosen.

Deep learning extends the simple linear regression model by using NNs, allowing to automatically find a suitable ϕ. More formally, they resemble a function $f(x)$ which approximates $f^*(x)$ where $f^*(x)$ is the true, unknown function. In general, NNs can approximate any function. See Hornik et al. (1989) and Kůrková (1992) for proofs on the approximation ability of NNs.

NNs consist of layers each with a possibly varying amount of neurons. The first layer, the input layer, takes the input $\mathbf{x}$ and computes each neuron $h_i \quad \forall i \in \{1, \ldots, j\}$ as

$$h_i^1 = f(\mathbf{w}_{1,i}^T \mathbf{x} + b_{1,i}),$$

where $\mathbf{w}_{1,i}$ are the weights and $b_{1,i}$ the constant bias. f is called activation function and is a non-linear, differentiable function such as the sigmoid func-

tion $\sigma(x) = \frac{1}{1+e^{-x}}$, the Rectified Linear Unit (ReLU)[3] $f(x) = \max(0, x)$ or the tangent hyperbolicus $tanh(x)$.

A (deep) NN additionally contains one or more hidden layers, where each layer takes as input the activation of the previous layer:

$$\mathbf{h}^l = f(W_l^T \mathbf{h}^{l-1} + \mathbf{b_l}). \tag{2}$$

Here, W_l is the matrix for hidden layer l, covering all weights for each neuron:

$$W = \begin{pmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nm} \end{pmatrix},$$

where $\mathbf{h}^l \in \mathbb{R}^m$ and $\mathbf{h}^{l-1} \in \mathbb{R}^n$ for one specific layer, since the amount of neurons per layer can vary. For an example NN, see fig. 3.

The output of the NN, or more specifically of the last layer, is the prediction $\mathbf{y}$ for the input sample data $\mathbf{x}$. During training, $\mathbf{y}$ is directly used in a loss function, which compares it to the ground truth, also known as label, of the corresponding input, class $\hat{y}$. For classification, the categorical cross-entropy loss

$$L_{CE}(\mathbf{y}, \hat{y}) = -\mathbf{y}[\hat{y}] + \log\left(\sum_{i=1}^{C} \exp(\mathbf{y}[i])\right) \tag{3}$$

is often used, where C denotes the number of classes. To update the weight of the NN, a gradient descent optimization algorithm is used. For example, the stochastic gradient descent updates each weight w of the NN with a learning rate η and batch size B as

$$w = w - \eta \cdot \frac{1}{B} \sum_{j=1}^{B} \nabla_w L_{CE}(\mathbf{y_j}, \hat{y}_j), \tag{4}$$

where batching is applied to make the stochastic gradient descent more stable and for faster computation.

[3]Although the ReLU is not differentiable at $x = 0$, the derivative is just set to be 0 or 1 for the scope of NNs, i.e. $f'(0) = 0$ or $f'(0) = 1$ respectively.

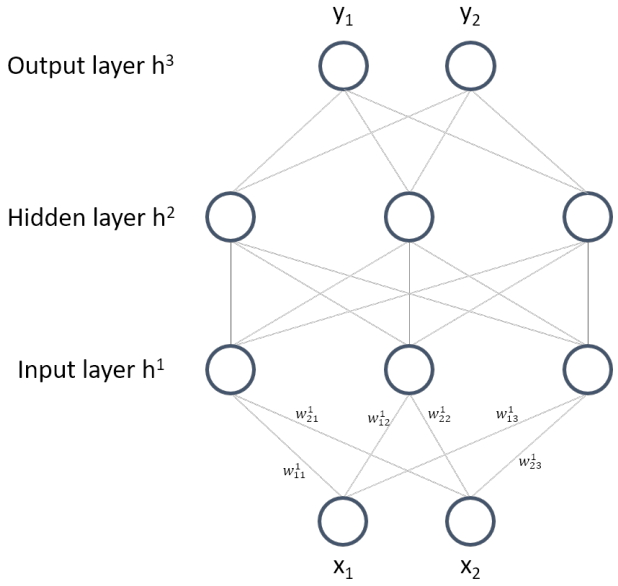

Figure 3: A Neural Network consisting of two inputs, the input layer and one hidden layer each with three neurons and two outputs in the output layer.

For this purpose, the derivatives of the loss function with respect to each weight w has to be computed using the chain rule:

$$\nabla_w L_{CE}(\mathbf{y}, \hat{y}) = \frac{\partial}{\partial w} L_{CE}(\mathbf{y}, \hat{y}) \tag{5}$$

The whole process is often referred to as *backpropagation* since the error from the loss function is propagated back to each neuron (Rumelhart et al., 1986).

It's worth mentioning that a trained NN should not reproduce the labels exactly because it would overfit, preventing the NN to generalize well on unseen data. Moreover, labels may contain noise and the true distribution is unknown anyway. For this purpose, some kind of regularization is added, while there are different methods to do so. The most basic regularization originates from linear regression, where a regularization term is applied to the model (Girosi et al., 1995). In NNs this can be implemented by adding an L^1 or L^2 norm of each weight multiplied by the regularization factor λ to

the loss before computing the gradients of all weights. Other regularization techniques include dropout, where neurons are randomly turned off during training (Srivastava et al., 2014; Wan et al., 2013). There is also focus in research on regularization in RNNs (Zaremba et al., 2014), an architecture which will be introduced in the next part.

3.2 Architectures

Besides the basic feed forward network, where the input is propagated forward only, several other architectures exist. One of them is the so-called Recurrent Neural Network (RNN). The idea is to have additional backward edges in the NN, such that computations in the current timestep are used in the next timestep. The input to the RNN is therefore a sequence, which is often time-dependent, and results in a sequence in the output. This enables its use in different scenarios, for example when the input is a full voice sample, actions from a dialog or even full sentences as in machine translation. Through this model, information is propagated to the next training sample, which allows to model correlations over a sequence or over time. The process is visualized in fig. 4

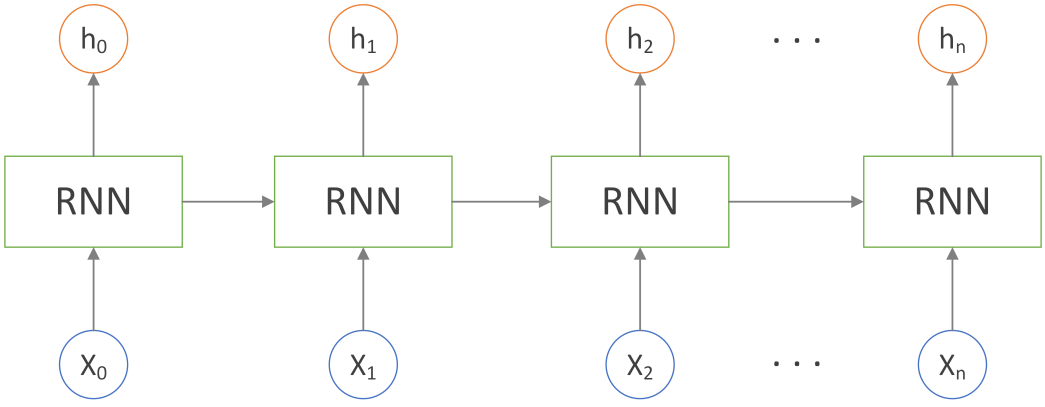

Figure 4: An RNN processes whole sequences and learns time dependent correlations. (adapted from Goodfellow et al. (2016))

A rather complex RNN has been presented by Hochreiter and Schmidhuber (1997), which aims to solve the problem of vanishing gradients for the weights in a standard RNN over long time. The LSTM computes the output vector $\mathbf{h}_t$ at timestep t as follows:

$$\mathbf{f}_t = \sigma_g(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + \mathbf{b}_f) \tag{6}$$

$$\mathbf{i}_t = \sigma_g(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + \mathbf{b}_i) \tag{7}$$

$$\mathbf{o}_t = \sigma_g(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + \mathbf{b}_o) \tag{8}$$

$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \sigma_c(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1} + \mathbf{b}_c) \tag{9}$$

$$\mathbf{h}_t = \mathbf{o}_t \circ \sigma_h(\mathbf{c}_t), \tag{10}$$

where $\circ$ denotes the element wise Hadamard product. The idea of LSTMs is to store information to be propagated over time in a memory cell $\mathbf{c}_t$, for which the input gate $\mathbf{i}_t$ controls the input to the cell (i.e. what parts of the input should be stored), the forget gate $\mathbf{f}_t$ determines which part of the cell from the previous timestep remains in the cell and the output gate $\mathbf{o}_t$ governs which part of the cell is used to compute the final result.

RNNs are also the main component of encoder-decoder architectures. An encoder RNN encodes all information from an input sequence into an internal state representation which is then used to generate a sequence using another RNN as decoder.

The main advantage of this architecture is to output a different length of sequence than used for the input, and will be used for this purpose in this thesis' contribution.

4 Resources

This section will provide an overview of the resources which were employed within this work.

4.1 PyDial

As this work depends on an available dialog system, the PyDial framework is used to provide the RL agent as well as an overall system to deliver and evaluate data.

PyDial (Ultes et al., 2017) is a benchmarking framework to train and evaluate RL agents within the context of an SDS. The authors defined six environments which differ in the domain, the input error, the user model and the system action masking. The domains covered in this thesis consist of Cambridge Restaurants (CR), San Francisco Restaurants (SFR) and Laptops (LAP). See table 1 for details on those domains.

Domain	Code	# constraint slots	# requests	# values
Cambridge Restaurants	CR	3	9	268
San Francisco Restaurants	SFR	6	11	636
Laptops	LAP	11	21	257

Table 1: The domains from PyDial (Ultes et al., 2017; p.5) used in this thesis. Although not explicitly mentioned, a user can also additionally inform about a name (e.g. after an offer in a request) adding an extra constraint slot to each domain.

The ontology for a domain not only specifies the requestable and informable slots, but also takes care of available actions for the system as well as for the user.

For simulation in PyDial, an agenda-based user simulator (cf. section 2.3.3) is already implemented. A set of parameters models the simulator's behavior and thus its friendliness.

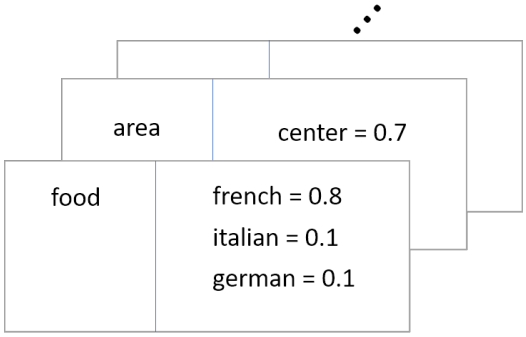

Figure 5: An example of a belief state as used in PyDial: a probability distribution for every slot yields the belief.

The input error, also referred to as semantic error rate, is simulated on top of the user action produced by the user simulator, while the masks limit the agent's possible actions in the current belief state to improve learning. The belief state is mainly a probability distribution over the possible values for each slot (see fig. 5 for an example).

	Env. 1			Env. 2			Env. 3			Env. 4			Env. 5			Env. 6		
task	T1.1	T1.2	T1.3	T2.1	T2.2	T2.3	T3.1	T3.2	T3.3	T4.1	T4.2	T4.3	T5.1	T5.2	T5.3	T6.1	T6.2	T6.3
Domain	CR	SFR	LAP	CR	SFR	LAP	CR	SFR	LAP	CR	SFR	LAP	CR	SFR	LAP	CR	SFR	LAP
SER	0%			0%			15%			15%			15%			30%		
Masks	On			Off			On			Off			On			On		
User	Standard			Standard			Standard			Standard			Unfriendly			Standard		

Table 2: The six environments defined by PyDial (Ultes et al., 2017; p.5)

An overview of those environments is given in table 2. As a simulation runs, an agent is trained on a specifiable amount of dialogs. Following an evaluation, the average rewards per turn, the success rate and the average turn length are reported.

4.2 PyTorch

The user simulator is written in Python and uses PyTorch[4] (Paszke et al., 2017) for the supervised learning part.

PyTorch is a library for python, making the power of torch available in this programming language. It integrates seemingly into the object orientated paradigm. Most important, PyTorch provides several loss functions and optimizers like Stochastic Gradient Descent (SGD) and Adam as well as different layers to build the architecture of a NN, also giving the possibility to add regularization.

It also takes care of the backpropagation, which results in the gradient of the weights which are used by an optimizer.

4.3 DSTC2 Dataset

The dataset of the second dialog state tracking (DSTC2) challenge[5] consists of a corpus of dialog between a user and a dialog manager. For the DSTC2 training and development set a handcrafted policy is used, while the test set only contains data from a dialog manager whose policy is learnt using RL. The voice input from the real users is converted to text using two different models. One of them artificially performs worse to have data with different noise available.

The DSTC2 dataset was initially released to fuel research on the dialog manager. Therefore, it focuses on training the dialog manager with its policy, but the user responses can yet be used to train a user simulator based on the system output. However, the transcriptions are defective and some dialogs miss any logical thinking. At least, they carry the naturalness of human subjects.

[4]https://pytorch.org/

[5]http://camdial.org/~mh521/dstc/

5 Neural-based User Simulation

This section will focus on the model's architecture as well as how it is trained and integrated into the PyDial framework.

5.1 Possible actions

As the output of the neural-based user simulator are actions on the intention-level, this part will focus on which actions exist and what they stand for. Table 3 shows the actions the model can produce and table 4 contains an overview of the actions which are considered in the input.

Action	**Arguments**	**Description**
null	–	actually indicating that there is no action
inform	slot-value pair	user informs about the value of the given slot, e.g. inform(food=french)
bye	–	user takes one's leave
request	slot	user requests the slot from the system, e.g. request(postcode)
ack	–	user takes note of the system's action
thankyou	–	user thanks the system for giving information
negate	–	user says "No."
hello	–	user greets the system
confirm	slot-value pair	user asks the system for confirmation of constraint, e.g. confirm(food=french)
reqalts	–	user requests alternative suggestions
affirm	–	user affirms
deny	slot-value pair	user denies the value for the given slot, e.g. (food=italian) often appears together with informing about the wanted value for this slot
repeat	–	user asks the system for repetition of its last action
reqmore	–	user asks for more information in general

Table 3: The actions the user simulator can produce.

5.2 Features

To model the context around the explicit goal representation, the idea from El Asri et al. (2016) is picked up and a feature vector $\mathbf{c}_t$ for each timestep t is manually crafted. This allows to implicitly keep track of the goal, and

Action	Arguments	Description
welcomemsg	–	system greets the user
offer	value	system makes a proposition (i.e. offers value as a venue), e.g. offer(name=la baguette)
inform	slot-value pair	system informs about the value of the given slot (of the suggested venue)
request	slot	system requests the slot from the user, e.g. request(food)
reqmore	–	system asks if the user wants more, e.g. if there are outstanding requests
canthelp	slot-value pairs	system tells that there are no venues for the listed constraints, e.g. canthelp(food=french), canthelp(area=north)
canthelp.exception	slot-value pair	the only venue(s) which match(es) the constraints, e.g. canthelp.exception(name=la baguette); only together with *canthelp* action and usually after the user requested alternative venues

Table 4: The possible system actions which are encoded in the context vector.

feed it into a NN. A goal is a list of constraints and requests, e.g. constraint(food=french), constraint(pricerange=expensive), request(name), request(addr). A constraint denotes a slot-value pair with inform action which shows up in the goal while an inform slot is generally a slot the user can inform about (during the course of the dialog). Since the feature vector models the context of the dialog, it is also called a context vector. It takes into consideration the state of the goal of the simulated user and the system's behavior. More precisely, $\mathbf{c}_t$ is a concatenation of the following binary vectors:

- Constraint status $const_t$
- Request status req_t
- Inconsistencies on value level made by the system after an offer $incon_t$
- Missing slots in offer $missing_t$

- Last system act $machine_t$

The vector $const_t$ keeps track about what the user informed so far. For this purpose, it is a binary vector with length n_i, where n_i is the number of informable slots (4 in our case: *food*, *pricerange*, *area*, *name*). It is initialized at the beginning of a dialog with 0 for each constraint in the goal and 1 for all other constraints. Whenever the user informs about a constraint, e.g. inform(food=french), the corresponding slot in $const_t$ is set to 1. A particular slot is only reset when the goal is changed (e.g. there is no venue for a slot, thus it has to be relaxed). Thus the idea is that this vector represents all inform actions needed to elucidate about the constraints in the goal.

The next vector, req_t, is similarly constructed, but keeps track of the requests in the goal instead of the constraints. Therefore it has length n_r, which is the number of request slots (9 in our case: *signature*, *addr*, *phone*, *description*, *postcode*, *food*, *pricerange*, *area*, *name*). Since the system should take care of informing about a request slot in the goal and a user's request is not guaranteed to be answered by the system, this vector is only set to 1 if the system informs about a request slot in the goal. It is initialized similar to $const_t$, that means 0 for all requests in the goal and 1 for all others. As a request depends on a venue (e.g. two restaurants usually do have a different address), it is not only reset on goal updates but also if the system offers a new venue.

One of the main parts to check whether the system understands the user's goal correctly or if it made errors (e.g. due to noise) is the vector modeling inconsistencies on the value level. This vector, $incon_t$, with length n_i is therefore initialized with zeros and set to 1 if the system mentions a slot-value pair which is in the goal but with a different value. Those actions from the system include *offer*, *confirm*, *inform*, *impl-conf* and *expl-conf*. Similar to the request vector, $incon_t$ is reset on a new offer from the system, but not on the first as all slot-value pairs the system informs about are considered to belong to the first offer before even proposing one. Additionally, whenever a

slot is set to 1, it is set to 0 in the constraint vector $const_t$. This reflects the matter of fact that the user has to inform about this slot again in order to rectify the system's belief.

Another vector also takes on inconsistencies made by the system, but those which are less harmful. For this purpose, $missing_t$ is a turn-wise vector with length n_i which is initially 0 and set to 1 for each constraint mentioned by the system in this turn's offer. It gives further information to the NN as some users want to confirm a missing but not wrongly understood (by the system) slot in the offer such that the offer does not violate the goal.

The last, and obviously also important information, is the last system action. It is encoded as a vector of size n_{ma}, where n_{ma} is the number of possible system actions, here 7. Table 4 lists those. Each of the actions in the last system's turn is set to 1 in $machine_t$ after initializing the vector with 0 each turn.

5.3 Model

The sequence of context vectors for each turn $\mathbf{c}_0, \ldots, \mathbf{c}_t$ is used as input for the NN at timestep t. Since the input is a sequence and it should be possible to output multiple actions, the encoder-decoder architecture is used for the neural-based user simulator with a one-layered, unidirectional LSTM in both cases. From eq. (6) it follows for a complete sequence

$$\mathbf{s}_t = \text{LSTM}(\mathbf{c}_0, \ldots, \mathbf{c}_t), \tag{11}$$

where $\mathbf{s}_t$ is the hidden state for turn t. The output and the cell state are neglected here since they won't be used in the further computation. The hidden state of the encoder RNN is then forwarded through a fully connected linear layer to produce the initial hidden state for the decoder RNN

$$\mathbf{h}_t = W_e^T \mathbf{s}_t + \mathbf{b}_e. \tag{12}$$

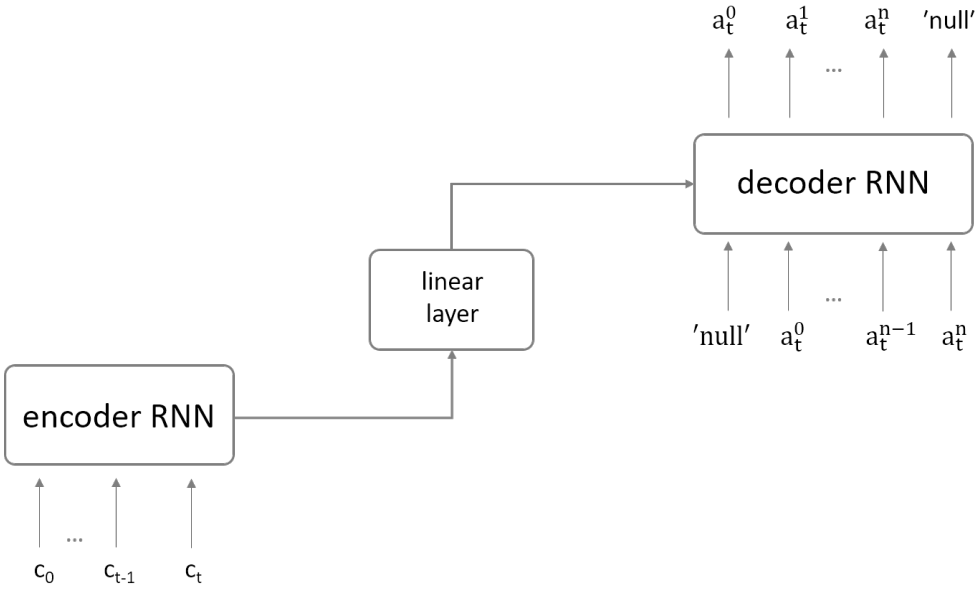

Figure 6: The user simulator model: The context history is encoded into a sequence of vector representations $(\mathbf{c}_0, \ldots, \mathbf{c}_t)$. After passing the encoder, it is used as input for the decoder which in turn outputs the actions $(a_t^0, \ldots, a_t^n)$ until the stop signal *null* is generated.

On top of the decoder, a fully connected linear layer maps to the action space whereafter the softmax is computed to produce a probability distribution, from which the actual output action is drawn (according to this distribution):

$$\mathbf{o}_t^n = W_d^T \cdot \text{LSTM}(\mathbf{a}_t^{n-1}, \mathbf{h}_t) + \mathbf{b}_d \tag{13}$$

$$\mathbf{a}_t^n = \frac{\exp(\mathbf{o}_t^n)}{\sum_i \exp(\mathbf{o}_t^n)_i} \tag{14}$$

Here, $\mathbf{a}_t^{-1}$ is the one-hot encoded null action.

This sampling adds variance in the output and also captures the different actions with the same condition as seen in the training data. For each timestep t, the decoder takes a start token, the *null* action, and produces action a_t^0 as first action. From there on, the last decoded action is used one-hot encoded as input in order to get the next action and so on. The decoder is run until a stop token is produced, which is the *null* action again. Figure 6 visualizes the whole process.

Mathematically speaking, each action a_t^n is a probability distribution conditioned on the initial decoder hidden state $\mathbf{h}_t$ and all actions $a_t^{-1}, \ldots, a_t^{n-1}$:

$$P(a_t^n \mid a_t^{n-1}, \ldots, a_t^{-1}, \mathbf{h}_t) \tag{15}$$

The probability of the whole sequence $a_t^n, a_t^{n-1}, \ldots, a_t^0$ is then

$$P(a_t^n, a_t^{n-1}, \ldots, a_t^0 \mid \mathbf{h}) = \prod_{i=0}^{n} P(a_t^i \mid a_t^{i-1}, \ldots, a_t^{-1}, \mathbf{h}). \tag{16}$$

As already written, for both the encoder and the decoder a one-layered, unidirectional LSTM was used. The model was trained with the categorical cross-entropy loss function from eq. (3) presented in section 3.1 in conjuction with SGD and a learning rate of 0.01. L^2-regularization was used with λ set to 0.001. For batching, 128 randomly, independent dialog turns were sampled, each of them containing the full context history.

During training, no teacher forcing (i.e. using the label as input for the decoder) was used as the computation time was still affordable. Since the

decoding step in the decoder RNN can possibly run forever (i.e. if no stop token is produced), the maximum decoding length was set to 5. This equals the maximum number of actions in all data used in this thesis.

5.4 Integration into PyDial

Since the user simulator should be used to train RL policies as described in the introduction (section 1.2), it has been integrated into the PyDial framework presented in section 4.1.

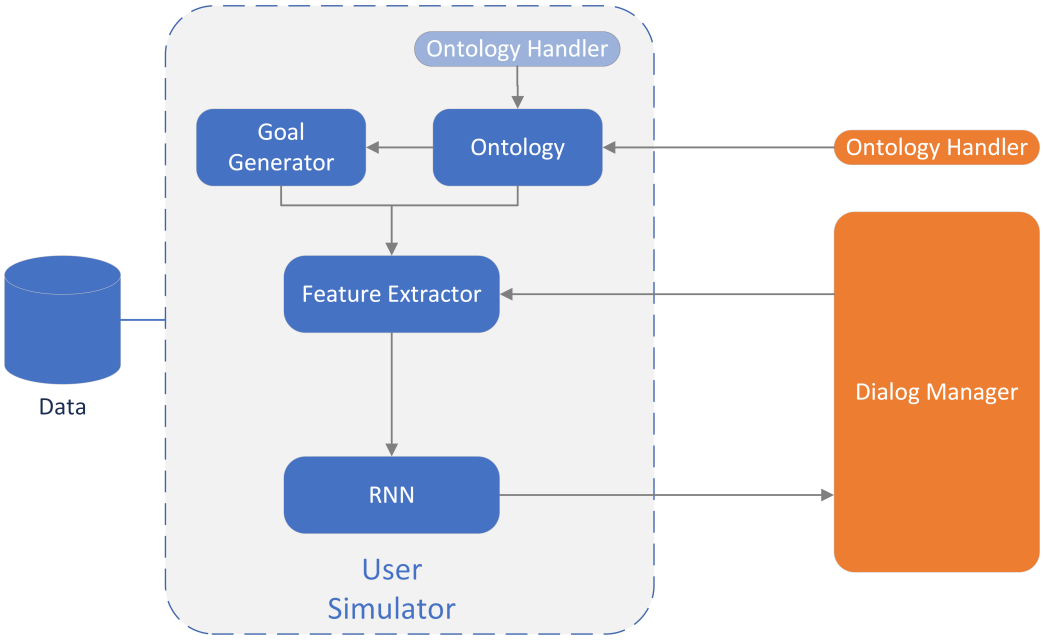

Figure 7: The user simulator integrated into PyDial: An optional Ontology Handler extracts the ontology from the training data, which fuels the ontology for the simulator. Otherwise PyDial's handler is used. The ontology is used to generate goals, and is used together with the goal and the last system turn to extract the features which are used as input for the RNN. The action is then issued to the dialog manager, which feeds the system answer back.

Figure 7 shows the full overview on a high level: The user simulator holds an ontology from the domain. This ontology is either provided by PyDial

directly, or optionally collected from the training data. Using the knowledge from the ontology, the goal generator randomly draws a goal at run time (i.e. when used in PyDial) at the beginning of a dialog. As described in section 5.2, the features are built using the goal, the last system action and the ontology for fixed mappings of actions and slots. After forwarding the feature vector through the rnn, the predicted user actions are handed to the DM. This module in turn uses the action as input, and produces the next system action (by running through the update of the belief state and the policy as already described). The system action is then used in the next turn in the user simulator again.

When training the user simulator, none of the parts provided by PyDial are used. Instead, all actions and goals for each dialog are collected from the data. While training, only the mappings from the ontology are used (which are mapped to actions on the intention-level when running with the DM).

6 Experiments & Results

Within this section, the previously introduced model is used in its field of application with the SDS. Therefore, this part differentiates between using synthetic and real data for training.

6.1 Synthetic Data

In a first approach, this section will consider to model the handcrafted simulator. Therefore, the neural-based user simulator will be trained on different synthetic data as well as compared to the handcrafted simulator, from which the data has been generated.

6.1.1 Separate environments

Following the idea of reproducing PyDial's handcrafted simulator first, this agenda-based simulator (results shown in table 5) is directly compared to the neural-based simulator (results shown in table 6) within PyDial. For this purpose, several dialogs from Pydial using the handcrafted user simulator and the handcrafted policy were generated. The environments 1, 3, 5 & 6 were used since 2 & 4 have the masks turned off which do only influence the policy in general (but not the handcrafted one). This implies that the generated data differs in noise and the user model (Standard vs. Unfriendly). The noise in this work only models action type misunderstanding as the action type is the only output from the model as presented in section 5.3. However the noise is not seen as ASR noise for the DM, but rather to model different users resulting in a varying behavior (so all actions will be reflected in the user's state).

For each of the 4 environments, 6000 training dialogs and 2000 testing dialogs have been collected. Using this corpus of labelled data, the same configurations (i.e. same policies and domains) for both simulators were used

to obtain the results in table 6 and table 5 respectively. Hyperparameters were taken from Casanueva et al. (2017). However those were optimized for the handcrafted simulator.

To generate the results, for each environment the corresponding data was used. Thus, environment 1 with the neural-based user simulator is represented by using the generated data (with the handcrafted simulator) from environment 1, i.e. without noise and using the standard user model. In contrast, environment 2 was gained by using the same data as for environment 1, but disabling any masks simplifying the policy w.r.t. action sampling. It should be noted that the semantic error noise has been set to 0 whenever using the RNN simulator within PyDial as the noise is inherent to the generated data (this was the core of the first idea of reproducing the handcrafted user simulator). Therefore the only change in the environments for the neural-based simulator is in the masks.

Another important fact is that, because of the deterministic ASR, some actions were never used by the handcrafted policy, namely *select* and *confirm*. As the RNN simulator was trained using this data, it has never seen those actions as input. To not increase the difficulty for the simulator due to the generalization to unseen actions (those weights could become 0 after training, or any other value), both actions were not used for any policy for the results gained with synthetic data.

For the CR domain, the RNN simulator gives comparable results to the handcrafted simulator for all environments when looking at the handcrafted policy. The average over all environments shows a reward of 12.1 and a success rate of 93.7% for the RNN simulator in comparison to the handcrafted one (12.5 and 96.4%). This indicates that the simulator models the behavior of the handcrafted simulated user for this domain. The other domains, however, degrade even faster with the neural-based user simulator. Those domains have a much larger state and action space, and the handcrafted policy has been probably tuned to achieve high results with the handcrafted simulator. However, due to its stochasticity this deep learning based user

		GP-Sarsa		DQN		A2C		eNAC		Handcrafted	
	Task	Suc.	Rew.	Suc.	Rew.	Suc.	Rew.	Suc.	Rew.	Suc.	Rew.
Env. 1	CR	99.6%	**13.7**	94.7%	12.9	97.4%	13.5	94.3%	12.1	100.0%	14.1
	SFR	97.7%	11.9	72.4%	7.8	76.1%	8.5	95.2%	**12.0**	98.6%	12.9
	LAP	88.4%	9.1	57.9%	5.0	76.2%	8.3	91.3%	**10.4**	97.6%	12.0
Env. 2	CR	98.6%	**13.4**	81.5%	8.7	23.8%	1.9	62.2%	5.0	99.9%	14.0
	SFR	95.5%	**12.0**	81.9%	8.8	5.0%	−1.2	75.8%	8.6	98.9%	13.0
	LAP	87.5%	**9.7**	50.2%	4.5	2.6%	−1.7	81.1%	9.0	97.3%	11.9
Env. 3	CR	97.3%	12.3	95.0%	**12.4**	88.3%	11.0	91.3%	11.2	96.8%	12.7
	SFR	88.9%	**8.9**	68.4%	6.2	62.8%	5.3	82.5%	8.6	88.7%	9.9
	LAP	73.9%	4.9	57.9%	3.9	54.9%	3.7	76.0%	**6.8**	83.3%	8.1
Env. 4	CR	94.2%	**11.9**	79.7%	8.8	5.7%	−1.1	38.6%	3.1	97.0%	12.7
	SFR	86.4%	**9.7**	75.5%	7.6	1.8%	−1.9	59.2%	4.6	89.2%	10.0
	LAP	52.9%	4.6	69.8%	6.0	2.4%	−1.5	78.5%	**8.2**	83.4%	8.1
Env. 5	CR	96.3%	**11.1**	92.7%	10.9	88.4%	10.0	93.6%	10.7	95.8%	11.5
	SFR	85.5%	**6.7**	61.1%	3.2	40.3%	0.1	78.2%	6.5	85.1%	7.5
	LAP	56.1%	−0.2	31.8%	−1.4	27.6%	−2.3	65.9%	**2.6**	77.1%	4.8
Env. 6	CR	89.8%	9.8	88.4%	**10.1**	76.8%	7.8	86.0%	9.3	89.1%	10.2
	SFR	76.2%	**5.4**	56.6%	2.9	48.3%	1.7	67.1%	4.9	74.4%	6.1
	LAP	51.4%	0.4	45.9%	0.9	34.2%	−0.8	58.6%	**2.7**	66.5%	4.0
Mean	CR	96.0%	**12.0**	88.7%	10.6	63.4%	7.2	77.7%	8.6	96.4%	12.5
	SFR	88.4%	**9.1**	69.3%	6.1	39.1%	2.1	76.3%	7.5	89.1%	9.9
	LAP	68.4%	4.7	52.3%	3.2	33.0%	1.0	75.2%	**6.6**	84.2%	8.1
	ALL	84.2%	**8.6**	70.1%	6.6	45.1%	3.4	76.4%	7.6	89.9%	10.2

Table 5: Success rates and rewards after training 4000 dialogs with the handcrafted simulator (averaged over 10 seeds).

simulator is able to generate new dialogs which will never occur with the handcrafted user simulator.

It is still remarkable that the GP-Sarsa policy performs better with the neural-based simulator in two domains (CR: reward of 12.7 vs. 12.0; SFR: reward of 9.6 vs. 9.1) when compared with the handcrafted simulator. Obviously, the GP-Sarsa algorithm can generalize better, even in larger state and action spaces as the average reward over all domains and environments is

	Task	GP-Sarsa Suc.	GP-Sarsa Rew.	DQN Suc.	DQN Rew.	A2C Suc.	A2C Rew.	eNAC Suc.	eNAC Rew.	Handcrafted Suc.	Handcrafted Rew.
Env. 1	CR	99.9%	**13.9**	48.8%	3.0	98.8%	13.7	67.4%	6.2	97.7%	13.5
	SFR	98.1%	**11.8**	20.3%	−3.1	34.3%	−0.7	87.6%	9.8	69.4%	6.9
	LAP	81.6%	**5.8**	12.9%	−4.2	41.7%	0.5	55.4%	2.3	57.6%	3.6
Env. 2	CR	98.0%	**13.5**	77.5%	8.8	25.6%	2.5	3.8%	−6.1	98.4%	13.6
	SFR	95.7%	**11.7**	60.0%	4.6	10.6%	−0.8	9.9%	−3.5	72.0%	7.4
	LAP	72.9%	**6.6**	20.6%	0.2	4.1%	−2.7	7.3%	−5.9	56.7%	3.3
Env. 3	CR	98.9%	**12.5**	49.3%	2.8	95.4%	12.2	78.3%	8.4	93.3%	12.0
	SFR	94.9%	**9.0**	18.2%	−3.8	33.0%	−1.6	77.4%	6.3	64.3%	5.2
	LAP	78.5%	**3.8**	9.9%	−4.9	30.0%	−2.1	44.9%	−0.3	43.8%	0.5
Env. 4	CR	98.1%	**12.7**	73.3%	6.8	7.3%	−0.3	6.3%	−4.3	92.3%	11.8
	SFR	89.9%	**9.3**	48.4%	3.5	1.6%	−1.5	17.1%	−4.7	65.7%	5.3
	LAP	23.2%	**1.0**	27.6%	−0.0	1.2%	−1.8	17.7%	−3.9	43.2%	0.4
Env. 5	CR	98.3%	**12.1**	26.5%	−1.7	88.8%	10.6	83.6%	9.0	95.0%	12.0
	SFR	95.7%	**8.8**	9.4%	−5.4	25.1%	−3.4	75.8%	5.3	63.9%	4.5
	LAP	72.3%	**2.7**	8.6%	−5.3	16.8%	−4.6	50.3%	−0.5	33.9%	−2.2
Env. 6	CR	96.1%	**11.2**	23.9%	−2.5	84.9%	9.3	84.4%	9.0	85.7%	9.9
	SFR	87.3%	**7.0**	5.5%	−6.0	15.6%	−5.0	71.7%	4.7	54.7%	2.7
	LAP	56.9%	**-0.6**	5.8%	−5.7	11.9%	−4.9	41.6%	−1.8	29.1%	−2.9
Mean	CR	98.2%	**12.7**	49.9%	2.9	66.8%	8.0	54.0%	3.7	93.7%	12.1
	SFR	93.6%	**9.6**	27.0%	−1.7	20.0%	−2.2	56.6%	3.0	65.0%	5.3
	LAP	64.2%	**3.2**	14.2%	−3.3	17.6%	−2.6	36.2%	−1.7	44.1%	0.5
	ALL	85.4%	**8.5**	30.4%	−0.7	34.8%	1.1	48.9%	1.7	67.6%	6.0

Table 6: Success rates and rewards after training 4000 dialogs with the RNN simulator trained on synthetic data (averaged over 10 seeds).

the largest with 8.5 (for the RNN simulator) and second largest with 8.6 (for the handcrafted simulator). GP-Sarsa is a highly sophisticated, Gaussian-Processes based algorithm extending the basic on-policy Sarsa algorithm. It is not parametric and thus does not suffer from instability in contrast to Deep Q Network (DQN), Advantage Actor Critic (A2C) and episodic Natural Actor Critic (eNAC). The latter policies show already an instable behavior with the handcrafted simulator (cf. performance from environment 1 and 2, where

all 3 policies perform worse, with the A2C policy gliding totally off except the DQN, which suddenly performs better in the second, more difficult environment for domain SFR). Therefore no conclusion can be drawn for the RNN user simulator w.r.t. those policies.

However it shall be in mind that the success rates and rewards do not reflect directly the user simulator's performance as it measures the quality of the policy, but the user simulator should help the policy achieving its goal. Without its support the policy can just make a guess. As stated above, the handcrafted policy gives a good hint that the RNN user simulator indeed approximates the handcrafted one as it is trained with data generated from this policy. At least one algorithm, GP-Sarsa, proves successful in those new environments suggesting the neural-based user simulator models a user's behavior semantically correct.

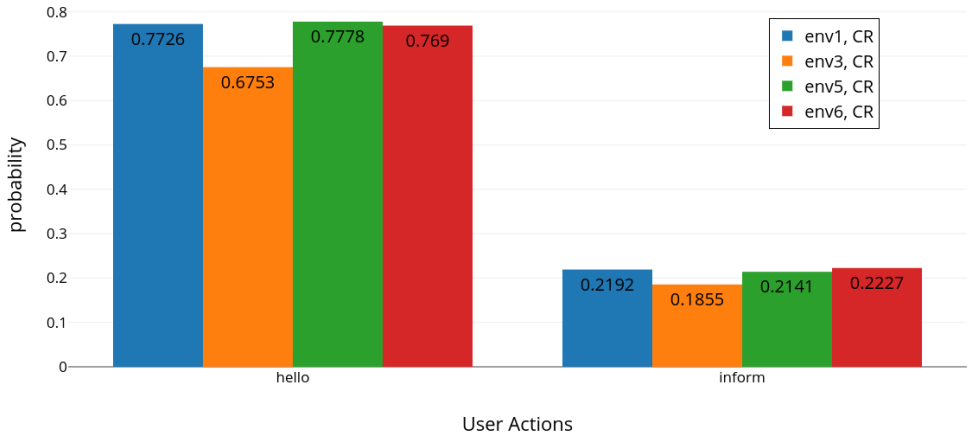

Figure 8: The probability distribution over user actions *hello* and *inform* after the system starts the dialog with a *welcomemsg* action. The other actions have a very small probability and are omitted for clarity.

Figure 8 shows the probability for the two most probable user actions from the RNN simulator after the system issued the *welcomemsg* action. For two datasets (env 1 & 6) the simulator exhibits the same probability as seen

Environment	Env. 1	Env. 3	Env. 5	Env. 6
Probability	76.98%	76.15%	84.25%	76.71%

Table 7: Probability for the user action *hello* at the start of a dialog as seen in the synthetic data from the CR domain and the listed environments.

in the training data (cf. table 7). The other environments are slightly under estimated.

The problem, that the RNN simulator trained on data from the hand-crafted simulator doesn't perform well in a real environment, remains, as the following dialog shows by way of example (GP-Sarsa trained for 4000 dialogs with environment 1, see first row of table 6):

SYSTEM: Welcome to the Cambridge restaurant dialogue system. How may I help you?

USER: Could you tell me the name of a chinese restaurant?

SYSTEM: What part of town do you have in mind?

USER: centre

SYSTEM: Would you like something in the cheap, moderate, or expensive price range?

USER: that doesn't matter

SYSTEM: shanghai family restaurant is a nice place, it serves chinese food , it is in the moderate price range and it is in the centre of town .

USER: Thanks, and what is the phone number?

SYSTEM: Can I help you with anything else?

USER: Yes, the phone number would be really awesome

SYSTEM: Can I help you with anything else?

USER: give me the phone number. Now.

SYSTEM: Can I help you with anything else?

USER: are you kidding?

SYSTEM: shanghai family restaurant is a nice place Their phone number is 01223 301761 .

USER: Thank you! And bye

SYSTEM: Thank you, goodbye.

However, the user finally reached their goal and the simulator is better than the pure handcrafted simulator as it can produce new dialogs and help the policy to explore other states. The following shows an example dialog using the same configuration (i.e. the first environment trained with the GP-Sarsa policy using 4000 iterations and the handcrafted simulator (first row of table 5):

SYSTEM: Welcome to the Cambridge restaurant dialogue system. How may I help you?

USER: Could you tell me the name of a chinese restaurant?

SYSTEM: What part of town do you have in mind?

USER: centre

SYSTEM: What kind of food would you like?

USER: chinese

SYSTEM: What kind of food would you like?

USER: I still want chinese food!

SYSTEM: Would you like something in the cheap, moderate, or expensive price range?

USER: that doesn't matter

SYSTEM: You are looking for a chinese venue right?

USER: that's correct

SYSTEM: Let me confirm, You are looking for a venue and you dont care about the price range right?

USER: yes, you're all right

SYSTEM: Let me confirm, You are looking for a venue and you dont care about the price range right?

USER: affirm

SYSTEM: Let me confirm, You are looking for a venue and you dont care about the price range right?

USER: okay I will break here if you don't offer a venue

SYSTEM: I am sorry but there is no place other than venue .

USER: are you sure? it must be!

SYSTEM: I am sorry but there is no place other than venue .

USER: Bye

SYSTEM: Thank you, goodbye.

6.1.2 Mixed environments

As it is crucial to the working of a dialog system to have one trained policy available for use in a real scenario, another approach in this thesis was to propose a new environment. This environment aggregates several other environments from PyDial, removing the need to choose a specific environment. In detail, all synthetic data from the previous part were used, giving a total of 24000 train dialogs and 8000 test dialogs. Their semantic error rate is in average 15%. However, the main purpose of this dataset is that it offers some dialogs without any noise. This improves the user simulator and eventually the trained policy, while it still allows to incorporate different user behavior

induced by noise, possibly boosting the performance of trained RL policies in real environments.

Having trained the neural-based user simulator on this new dataset yielding the new environment 7, the results from PyDial are shown in table 8. For a better comparison, table 9 shows only the results of the environments with data used for environment 7. The mean in the latter corresponds to environment 1 (first 3 rows) of table 8, while environment 2 (rows 4 to 6) uses the same data, but applies masking for the policy action sampling.

		GP-Sarsa		DQN		A2C		eNAC		Handcrafted	
	Task	Suc.	Rew.	Suc.	Rew.	Suc.	Rew.	Suc.	Rew.	Suc.	Rew.
Env. 1	CR	99.3%	13.1	57.3%	4.1	98.0%	**13.2**	73.8%	7.2	96.6%	12.7
	SFR	94.4%	**9.2**	12.3%	−4.9	32.8%	−1.8	76.3%	6.1	63.7%	4.8
	LAP	64.4%	**2.1**	11.5%	−4.7	18.8%	−3.8	34.9%	−2.7	38.9%	−0.8
Env. 2	CR	98.4%	**13.1**	82.4%	9.2	22.0%	1.5	2.5%	−6.5	96.2%	12.7
	SFR	89.2%	**9.4**	49.0%	3.1	9.6%	−0.6	6.0%	−5.0	65.1%	5.1
	LAP	10.3%	0.2	44.1%	**3.4**	4.4%	−2.1	7.6%	−5.0	40.7%	−0.3

Table 8: Success rates and rewards after training 4000 dialogs with RNN simulator and environment 7 (averaged over 10 seeds).

For the CR domain it turns out that environment 7 actually helps in the majority of all tested policies. The average reward increase for the trainable policies is 1.175. For the other domains the average reward decreases by 0.2 (SFR) and 0.675 (LAP). As the CR domain is much smaller than the others, it learns a good policy much faster. The circumstance that the policies' hyperparameters were not tuned probably absorbs any benefit in those domains.

6.2 Real Data

To train the policies such that they may help real users, a user simulator needs to be trained on real data. The DSTC2 described in section 4.3 provides such dialogs. However there are goal changes within those dialogs, because the

		GP-Sarsa		DQN		A2C		eNAC		Handcrafted	
	Task	Suc.	Rew.	Suc.	Rew.	Suc.	Rew.	Suc.	Rew.	Suc.	Rew.
Env. 1	CR	99.7%	**13.7**	48.8%	3.0	98.5%	13.6	77.0%	8.6	98.3%	13.6
	SFR	98.1%	**11.8**	20.3%	−3.1	51.9%	2.5	87.6%	9.8	73.7%	7.7
	LAP	81.6%	**5.8**	17.1%	−3.4	41.7%	0.5	55.4%	2.3	57.6%	3.6
Env. 3	CR	98.9%	**12.5**	49.3%	2.8	95.4%	12.2	78.3%	8.4	93.3%	12.0
	SFR	94.9%	**9.0**	18.2%	−3.8	33.0%	−1.6	77.4%	6.3	64.3%	5.2
	LAP	78.5%	**3.8**	9.9%	−4.9	30.0%	−2.1	44.9%	−0.3	43.8%	0.5
Env. 5	CR	98.3%	**12.1**	26.5%	−1.7	88.8%	10.6	83.6%	9.0	95.0%	12.0
	SFR	95.7%	**8.8**	9.4%	−5.4	25.1%	−3.4	75.8%	5.3	63.9%	4.5
	LAP	72.3%	**2.7**	8.6%	−5.3	16.8%	−4.6	50.3%	−0.5	33.9%	−2.2
Env. 6	CR	96.1%	**11.2**	23.9%	−2.5	84.9%	9.3	84.4%	9.0	85.7%	9.9
	SFR	87.3%	**7.0**	5.5%	−6.0	15.6%	−5.0	71.7%	4.7	54.7%	2.7
	LAP	56.9%	**-0.6**	5.8%	−5.7	11.9%	−4.9	41.6%	−1.8	29.1%	−2.9
Mean	CR	98.3%	**12.4**	37.1%	0.4	91.9%	11.4	80.8%	8.7	93.1%	11.9
	SFR	94.0%	**9.2**	13.4%	−4.6	31.4%	−1.9	78.1%	6.5	64.2%	5.0
	LAP	72.3%	**2.9**	10.3%	−4.8	25.1%	−2.8	48.1%	−0.1	41.1%	−0.3
	ALL	88.2%	**8.2**	20.3%	−3.0	49.5%	2.3	69.0%	5.1	66.1%	5.6

Table 9: Success rates and rewards after training 4000 dialogs with RNN simulator trained on synthetic data (averaged over 10 seeds).

users were asked to test a specific constraint first, and alter this constraint if there is no venue with the current goal (i.e. choose a different value). When training on this data, those goal changes will be reflected such that the actions are drawn consistently with the new goal. For this purpose, the eventual goal of the user - as represented in the data - is used as initial goal. Whenever the user informs about a different value for a constraint, this value is considered to be part of the new goal.

After training for 200 epochs on this data, the user simulator reached an F-score of 54,21% (cf. table 10), which is much higher than for the bi-gram (9%) and agenda-based model (18%) (El Asri et al., 2016). Since the RNN simulator only outputs action types, it cannot be directly compared to the performance of the others.

Dataset	**Bi-gram**	**Agenda-based**	**RNN**
DSTC2 Test	0.09^{+}	0.18^{+}	0.5421*

Table 10: F-score for different user models.

$^{+}$taken from El Asri et al. (2016) *only action type

		GP-Sarsa		DQN		A2C		eNAC		Handcrafted	
Task		Suc.	Rew.	Suc.	Rew.	Suc.	Rew.	Suc.	Rew.	Suc.	Rew.
Env. 1	CR	97.6%	**13.8**	62.9%	6.4	96.7%	13.7	90.7%	12.1	88.0%	12.3
Env. 2	CR	95.1%	**13.6**	69.5%	8.5	43.4%	5.3	34.0%	−0.5	88.1%	12.4
Mean	CR	96.4%	**13.7**	66.2%	7.5	70.1%	9.5	62.4%	5.8	88.1%	12.3

Table 11: Success rates and rewards after training 4000 dialogs with RNN simulator trained on DSTC2 (averaged over 10 seeds).

The performance of this user simulator (based on the DSTC2 dataset) within PyDial is shown in table 11. Since the handcrafted policy does not learn from data, it is only affected by the randomness of the random generator. In other words, different random generator seeds are the only part making up a difference. Even the parametric algorithms could prove useful, with rewards ranging between 6.4 and 13.7 for environment 1 and between -0.5 and 8.5 for environment 2. GP-Sarsa achieves again the highest rewards, indicating the user simulator helps in achieving the goal (at least in a way the RL algorithm can exploit). The following sample dialog shows a notable improvement to the handcrafted simulator and to the RNN model trained on synthetic data from the handcrafted simulator:

SYSTEM: Welcome to the Cambridge restaurant dialogue system. How may I help you?

USER: Could you tell me the name of a chinese restaurant?

SYSTEM: Would you like something in the cheap, moderate, or expensive price range?

USER: moderate is just fine

SYSTEM: What part of town do you have in mind?

USER: centre

SYSTEM: shanghai family restaurant is a nice place, it serves chinese food , it is in the moderate price range and it is in the centre of town .

USER: What is their address and phone number?

SYSTEM: shanghai family restaurant is a nice place, Their phone number is 01223 301761 and Their address is 39 burleigh street city centre .

USER: Thank you very much, good bye

SYSTEM: Thank you, goodbye.

Although this is a rather simple domain, the policy asks the user in a straightforward fashion for their constraints without wasting unnecessary turns. Using this simulator, policies are assumed to perform much better in a real environment.

7 Conclusion

Nowadays, SDSs are widely used and the emergence of RL as well as research in this topic drives intelligent, self-learning and adaptive SDSs in a variety of domains. Their potential is obvious: A user gets easy access to a database or may book a flight, wherever they may use their voice or text to interact with a dialog system.

There is a lot focus in research on the policy, but the user simulator is another crucial part of the whole system. It allows to train and evaluate different policies without the need for human subjects, or at least much less. This saves time, money and effort and fuels research on applying RL algorithms in this field.

Previous user simulators were based on handcrafted rules, or extracted parameters for stochastic models from data. NNs have gained attention in different fields, as they exhibit universal function approximators and allow to automatically correlate data. Furthermore, RNNs allow to process whole sequences, and thus enable to parse a dialog turn by turn, thereby emitting multiple (user) actions.

Within this work, a RNN based user simulator has been implemented and evaluated in a benchmarking environment. PyDial provides several environments for this purpose, from which synthetic data have been generated. In the first part, the neural-based simulator was shown to be successfully trained on this data, therefore learning the handcrafted simulator's behavior. It does perform similar to the handcrafted simulator, but benefits from its ability to explore new dialogs such that it can prove useful in a real environment.

Since several environments are not viable for the later use of a trained policy in a real environment, a new environment was proposed to combine some of PyDial's environments. For this purpose, the simulated noise is seen as a varying behavior from different users.

As a last work, and as a step to achieve more realistic policies, the RNN

simulator was trained on the DSTC2. It does not only give neat quantitative results, but also suggests a notable improvement when the trained policy is deployed in a real environment, where humans interact with the dialog system.

However there are currently some limitations on the model: It only predicts the type of an action, which limits the ability to model the user as seen in the data. Furthermore, it needs more manual effort when the action type is mapped to an action on the intention-level. Also, the last system actions have to be taken into account. For example, a user's *deny* action needs information on the mentioned slots in an appropriate action from the system. The model's dependency on a specific domain furthermore makes it less versatile.

Finally, the evaluation currently still needs real subjects. This makes it hard to evaluate multiple policies, trained with different data and with different parameters in a variety of domains.

8 Future Work

As already stated, an evaluation of a policy with real subjects would also allow to draw implications about the performance of the used simulator. A good and as realistic as possible user simulator is crucial for the success of RL policies. Therefore a user study where the RNN simulator trained on real data (DSTC2 in this case) is evaluated and compared to the agenda-based simulator would finalize this project. This could be implemented in a setting where the participants of the study directly interact with the user simulator (i.e. play the role of the system) and try to figure out the simulated user's goal. Another option would be to train one or more policies on both simulators, and let the users perform tasks, thereby also evaluating the policy. The latter approach makes it harder to distinguish between the additional benefits exposed by the user simulator and the policy. On the other hand, it is probably easier for humans to rate a dialog system as it is more intuitive.

Another work could also focus on using other, more appropriate data like the Wizard of Oz (WOZ) 2.0 dataset or gather data in a WOZ fashion. This would yield dialogs in a more sophisticated language. Crowdsourcing is to be taken into consideration when it turns to collecting data or actually evaluating results. Jurčíček et al. (2011) show that such an approach gives reliable results comparable to user studies with local subjects (at least for evaluation). However it would also be interesting to make the model domain independent, thus removing the need to collect data for each domain separately.

The limitation to output only action types could be attenuated by modeling the actions on a more fine-grained level which also includes the slots similar to El Asri et al. (2016). For example, instead of one *inform* action, there would exist a few actions like *inform_food*, *inform_area* and so on. Further research aiming at modeling actions on the finest level, where the values are also captured, is also conceivable.

Since there is still a lot of manual effort involved in extracting the features,

future work could also focus on using more advanced NN architectures. It should allow to use the action on intention- or even on text-level as input while still sticking to an explicit goal. In total, the input and the output part of the user simulator can be improved.

Of course, the contribution of the user simulator depends also on the utilized RL policy, and there are today more sophisticated algorithms. Those should be examined too. In general, the presented neural-based user simulator can be used as a basis for future research in user simulator supported SDSs.

Bibliography

H C Bunt. Rules for the interpretation, evaluation and generation of dialogue acts. annualprog(4):99–107, 1981a.

Harry Bunt. Conversational principles in question-answer dialogues. pages 119–141, 1981b.

Iñigo Casanueva, Paweł Budzianowski, Pei-Hao Su, Nikola Mrkšić, Tsung-Hsien Wen, Stefan Ultes, Lina Rojas-Barahona, Steve Young, and Milica Gašić. A Benchmarking Environment for Reinforcement Learning Based Task Oriented Dialogue Management. 2017.

Senthilkumar Chandramohan, Matthieu Geist, Fabrice Lefevre, and Olivier Pietquin. User Simulation in Dialogue Systems Using Inverse Reinforcement Learning. page 4, 2011.

Lucie Daubigney, Matthieu Geist, Senthilkumar Chandramohan, and Olivier Pietquin. A comprehensive reinforcement learning framework for dialogue management optimization. 6(8):891–902, 2012. ISSN 19324553. doi: 10.1109/JSTSP.2012.2229257.

Wieland Eckert, Esther Levin, and Roberto Pieraccini. User Modeling For Spoken Dialogue System. page 8, 1997.

Layla El Asri, Jing He, and Kaheer Suleman. A sequence-to-sequence model for user simulation in spoken dialogue systems. 08-12-Sept:1151–1155, 2016. ISSN 19909772. doi: 10.21437/Interspeech.2016-1175.

M. Gašić, M. Henderson, B. Thomson, P. Tsiakoulis, and S. Young. Policy optimisation of POMDP-based dialogue systems without state space compression. (2):31–36, 2012. doi: 10.1109/SLT.2012.6424165.

Kallirroi Georgila, James Henderson, and Oliver Lemon. Learning User Simulations for Information State Update Dialogue Systems. page 4, 2005.

Kallirroi Georgila, James Henderson, and Oliver Lemon. User Simulation for Spoken Dialogue Systems: Learning and Evaluation. page 4, 2006.

Federico Girosi, Michael Jones, and Tomaso Poggio. Regularization theory and neural networks architectures. 7(2):219–269, 1995.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep Learning*, volume 1. MIT press Cambridge, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. 9(8): 1735–1780, 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. 2(5):359–366, 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90020-8.

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

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement Learning: A Survey. 4:237–285, 1996. ISSN 1076-9757. doi: 10.1613/jair.301.

Simon Keizer, Milica Gašić, Filip Jurčíček, François Mairesse, Blaise Thomson, Kai Yu, and Steve Young. 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, 2010.

Věra Kůrková. Kolmogorov's theorem and multilayer neural networks. 5(3): 501–506, 1992. ISSN 0893-6080. doi: 10.1016/0893-6080(92)90012-8.

E Levin, R Pieraccini, and W Eckert. Learning dialogue strategies within the markov decision process framework. pages 72–79, 1997.

Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz. End-to-end task-completion neural dialogue systems. 2017.

Bing Liu and Ian Lane. An End-to-End Trainable Neural Network Model with Belief Tracking for Task-Oriented Dialog. pages 2506–2510, 2017a. doi: 10.21437/Interspeech.2017-1326.

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

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

Olivier Pietquin and Helen Hastie. A survey on metrics for the evaluation of user simulations. 28(1):59–73, 2013. ISSN 0269-8889, 1469-8005. doi: 10.1017/S0269888912000343.

Olivier Pietquin, Matthieu Geist, Senthilkumar Chandramohan, and Hervé Frezza-Buet. Sample-efficient batch reinforcement learning for dialogue management optimization. 7(3):1–21, 2011. ISSN 15504875. doi: 10.1145/1966407.1966412.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. 323(6088):533–536, 1986. ISSN 1476-4687. doi: 10.1038/323533a0.

Jost Schatzmann and Steve Young. The hidden agenda user simulation model. 17(4):733–747, 2009.

Jost Schatzmann, Karl Weilhammer, MATT Stuttle, Matt, and Steve Young. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. 21(02):97, 2006. ISSN 0269-8889. doi: 10.1017/S0269888906000944.

Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. 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*, NAACL-Short '07, pages 149–152. Association for Computational Linguistics, 2007a.

Jost Schatzmann, Blaise Thomson, and Steve Young. Statistical user simulation with a hidden agenda. pages 273–282, 2007b. doi: 8577807053.

Konrad Scheffler and Steve Young. Automatic Learning of Dialogue Strategy Using Dialogue Simulation and Reinforcement Learning. In *Proceedings of the Second International Conference on Human Language Technology Research*, HLT '02, pages 12–19. Morgan Kaufmann Publishers Inc., 2002.

John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge university press, 1969.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. 15(1):1929–1958, 2014.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*, volume 135. MIT press Cambridge, 1998.

Stefan Ultes, Lina M. Rojas Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Iñigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, Milica Gasic, and Steve Young. PyDial: A Multi-domain Statistical Dialogue System Toolkit. pages 73–78, 2017. doi: 10.18653/v1/P17-4013.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of Neural Networks using DropConnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. 2015.

Jason Williams, Antoine Raux, and Matthew Henderson. The dialog state tracking challenge series: A review. 7(3):4–33, 2016.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent Neural Network Regularization. 2014.

Erklärung (Statement of Authorship)

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein. [6]

Stuttgart, den 30.11.2018

(Maximilian Schmidt)

[6] Non-binding translation for convenience: I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination pro cedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.