Neural User Simulation for Corpus-based Policy Optimisation for Spoken Dialogue Systems

Florian L. Kreyssig, Iñigo Casanueva
Paweł Budzianowski and **Milica Gašić**
Cambridge University Engineering Department,
Trumpington Street, Cambridge, CB2 1PZ, UK
{flk24,ic340,pfb30,mg436}@cam.ac.uk

Abstract

User Simulators are one of the major tools that enable offline training of task-oriented dialogue systems. For this task the Agenda-Based User Simulator (ABUS) is often used. The ABUS is based on hand-crafted rules and its output is in semantic form. Issues arise from both properties such as limited diversity and the inability to interface a text-level belief tracker. This paper introduces the Neural User Simulator (NUS) whose behaviour is learned from a corpus and which generates natural language, hence needing a less labelled dataset than simulators generating a semantic output. In comparison to much of the past work on this topic, which evaluates user simulators on corpus-based metrics, we use the NUS to train the policy of a reinforcement learning based Spoken Dialogue System. The NUS is compared 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.

1 Introduction

Spoken Dialogue Systems (SDS) allow human-computer interaction using natural speech. Task-oriented dialogue systems, the focus of this work, help users achieve goals such as finding restaurants or booking flights (Young et al., 2013).

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 (RL) problem (Young et al., 2013; Roy et al., 2000; Williams and Young, 2007; Gašić and Young, 2014). In this framework, the system learns by a *trial and error* process governed by a *reward function*. User Simulators can be used to train the policy of a *dialogue manager* (DM) without real user interactions. Furthermore, they allow an unlimited number of dialogues to be created with each dialogue being faster than a dialogue with a human.

In this paper the Neural User Simulator (NUS) is introduced which outputs natural language and whose 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 based 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 and a decoder RNN which outputs natural language. Furthermore, the NUS generates its own goal and possibly changes it during a dialogue. This allows the model to be deployed for training more sophisticated DM policies. To achieve this, a method is proposed that transforms the goal-labels of the used dataset (DSTC2) into labels whose behaviour can be replicated during deployment.

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 policy of a reinforcement learning based SDS. In order to evaluate the NUS, an Agenda-Based User-Simulator (ABUS) (Schatzmann et al., 2007) is used to train another policy. The two 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 other. Furthermore, both trained policies are tested on real users. On both evaluation tasks the NUS outperforms the ABUS, which is currently one of the most popular off-line training tools for reinforcement learning based Spoken Dialogue Systems (Koo et al., 2015; Fatemi et al., 2016; Chen et al., 2017; Chang et al., 2017; Casanueva et al., 2018; Weisz et al., 2018; Shah et al., 2018).

The remainder of this paper is organised as follows. Section 2 briefly describes task-oriented dialogue. Section 3 describes the motivation for the NUS and discusses related work. Section 4 explains the structure of the NUS, how it is trained and how it is deployed for training a DM's policy. Sections 5 and 6 present the experimental setup and results. Finally, Section 7 gives conclusions.

2 Task-Oriented Dialogue

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).

Using RL the DM is trained to act such that is maximises the cumulative future reward. The process by which the DM chooses its next action is called its *policy*. 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.

3 Motivation and Related Work

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 impractical. Furthermore, a 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 over and over again.

A problem with learning directly from recorded dialogue corpora is that the state space that was visited during the collection of the data is limited; the size of the recorded corpus usually falls short of the requirements for training a statistical DM. However, even if the size of the corpus is large enough the optimal dialogue strategy is likely not to be contained within it.

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 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's policy. The DM's policy is optimised using the feedback given by either the user simulator or 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.

Most user-simulators work on the level of user semantics. These usually consist of a *user dialogue act* (e.g. inform, or request) and a corresponding slot-value pair. 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 take the full dialogue history into account 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. However, developing the graph structure requires extensive domain-specific knowledge. Pietquin and Dutoit (2006) combined features from Scheffler and Young's work with Eckert's Model, by conditioning a set of probabilities on an explicit representation of the user goal and memory. 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 not 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. The 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 (Schatzmann et al., 2007), which represents the user state elegantly as a stack of necessary user actions, called the *agenda*. The mechanism that generates the user response and updates the agenda does not require any data, though it can be improved using data. The model is conditioned on a goal for which it has update rules in case the dialogue system expresses that it cannot fulfil the goal. El Asri et al. (2016) modelled user simulation as a sequence-to-sequence task. 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 outperformed the ABUS on statistical metrics, which is not surprising given that it was trained by optimising a statistical metric and the ABUS was not.

The aforementioned work focuses on user simulation at the semantic level. Multiple issues arise from this approach. Firstly, annotating the user-response with the correct semantics is costly. More data could be collected, if the US were to output natural language. Secondly, research suggests that the two modules of an SDS performing Spoken Language Understanding (SLU) and belief tracking 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 work uses, systems which used no external SLU module outperformed all systems that only used an external SLU Module[1]. Training the policy of a DM in a simulated environment, when also using a joint system for SLU 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. 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. A word-level user simulator was also used in (Li et al., 2017) where it was built by augmenting the ABUS with a natural language generator.

[1]The best-performing models used both.

4 Neural User Simulator

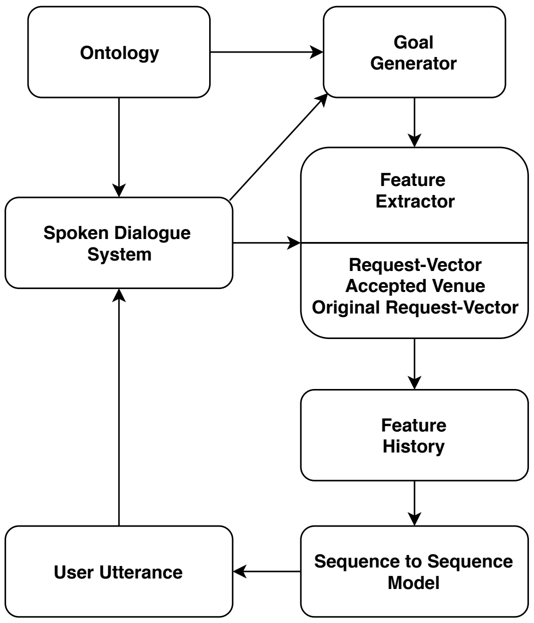

Figure 1: General Architecture of the Neural User Simulator. The System Output is passed to the Feature Extractor. It generates a new feature vector that is appended to the Feature History, which is passed to the sequence-to-sequence model to produce the user utterance. At the start of the dialogue the Goal Generator generates a goal, which might change during the course of the dialogue.

An overview of the NUS is given in Figure 1. At the start of a dialogue a random goal G_0 is generated by the *Goal Generator*. 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. 2), which will generate the user's length n_T utterance $\mathbf{u}_T = w_0 ... w_{n_T}$. As in Figure 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 goal will be altered by the Goal Generator.

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 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.

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 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 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 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 represen-

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 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.

tation 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 utterances of a user in response to `request(food)` and `request(area)` are often very different.

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 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 2) consists of an RNN encoder, followed by a fully-connect layer and an RNN decoder. An RNN can be defined as:

$$(\mathbf{h}_t, \mathbf{s}_t) = \text{RNN}(\mathbf{x}_t, \mathbf{s}_{t-1}) \quad (1)$$

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 $\mathbf{s}_t$. A specific RNN-design is usually defined using matrix multiplications, element-wise additions and multiplications as well as element-wise non-linear functions. There are a plethora of different RNN architectures that 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 exact LSTM version used in this work uses a forget gate without bias and does not use peep-holes.

The first RNN (shown as white blocks in Fig. 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:

$$\mathbf{p}_T = W_p \mathbf{h}_T^E + \mathbf{b}_p \quad (2)$$

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:

$$P(\mathbf{u}\,|\,\mathbf{p}) = P(w_0\,|\,\mathbf{p}) \prod_{t=1}^{L} P(w_t\,|\,w_{t-1}...w_0, \mathbf{p}) \quad (3)$$

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

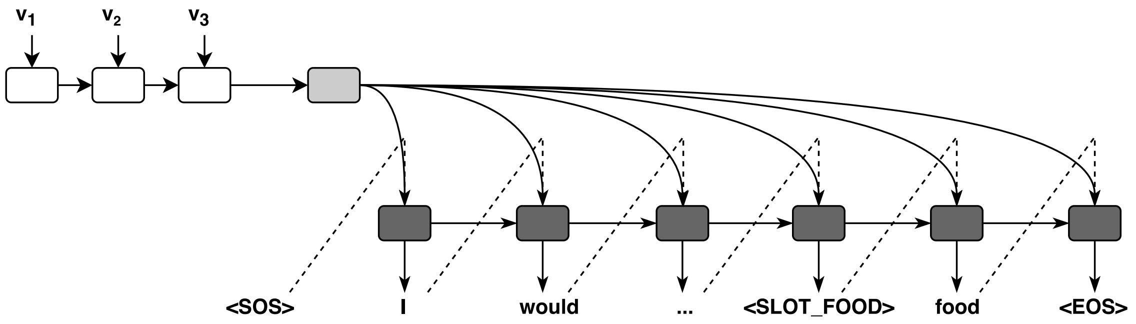

Figure 2: Sequence-To-Sequence model of the Neural User Simulator. Here, the NUS is generating the user response to the third system output. The white, light-grey and dark blocks represent the RNN encoder, a fully-connected layer and the RNN decoder respectively. The previous output of the decoder is passed to its input for the next time-step. $\mathbf{v}_{3:1}$ are the first three feature vectors (see Sec. 4.2).

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:

$$P(w_t \mid w_{t-1}...w_0,\mathbf{p}) = SM(W_w \mathbf{h}_t^D + \mathbf{b}_w) \quad (4)$$

$$w_t \sim P(w_t \mid w_{t-1}...w_0, \mathbf{p}) \quad (5)$$

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 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, for the NUS to always give the same response in the same context is not realistic. 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 2 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:

$$\mathcal{L} = \sum_{n=1}^{N} \log P(w_0 \,|\, \mathbf{p}) \sum_{t=1}^{L_n} \log P(w_t \,|\, w_{t-1:0}, \mathbf{p}) \quad (6)$$

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.

The manual transcriptions of the DSTC2 training set (not the ASR output) were used to train the sequence-to-sequence model. Since the transcriptions were done manually they contained spelling errors. These were manually corrected to ensure proper delexicalization. Some dialogues were discarded due to transcriptions 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 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* method of evaluating the policies that were obtained by training with each US.

5.1 Training

All dialogue policies were trained with the PyDial toolkit (Ultes et al., 2017), by interacting with either the NUS or ABUS. 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, its output is parsed using PyDial's regular expression based semantic decoder. The policies were trained for 4000 dialogues.

5.2 Testing with a simulated user

In Schatzmann 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 model still performs well when tested on poor user models. 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 our results we implement the same for the NUS by re-sampling a sentence if the new goal is not mentioned.

5.3 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. For the test on human users, 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. 6.1. 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.

6 Experimental Results

6.1 Cross-Model Evaluation

Table 2 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 94.0% and of 96.6% when tested on the ABUS and the NUS, respectively. By comparison, the policies trained with the ABUS achieved average SRs of 99.5% and 45.5% respectively. Thus, training with the NUS leads to policies that can perform well on both USs, which is not the case for training with the ABUS. Furthermore, the best SRs when tested on the ABUS are similar at 99.9% (ABUS) and 99.8% (NUS). When tested on the NUS the best SRs were 71.5% (ABUS) and

Train. Sim.	Eval. Sim.			
	NUS		ABUS	
	Rew.	Suc.	Rew.	Suc.
NUS-best	13.0	98.0$^{\mathcal{N}_1}$	13.3	99.8
ABUS-best	1.53	71.5$^{\mathcal{A}_1}$	13.8	99.9$^{\mathcal{A}_2}$
NUS-avg	12.4	96.6	11.2	94.0
ABUS-avg	-7.6	45.5	13.5	99.5

Table 2: Results for policies trained for 4000 dialogues on NUS and ABUS when tested on both USs for 1000 dialogues. Five policies with different initialisations were trained for each US. Both average and best results are shown.

Train. Sim.	Eval. Sim.			
	NUS		ABUS	
	Rew.	Suc.	Rew.	Suc.
NUS-best	12.2	95.9	13.9	99.9$^{\mathcal{N}_2}$
ABUS-best	-4.0	54.8	13.2	99.0
NUS-avg	12.0	95.4	12.2	97.3
ABUS-avg	-9.48	42.3	12.8	98.4

Table 3: As Table 2 but trained for 1000 dialogues.

98.0% (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.

Of the five policies, for each US, the policy performing best on the NUS was not the best performing policy on the ABUS. This could indicate that the policy "overfits" to a particular user simulator. Overfitting usually manifests itself in worse results 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 3. 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. This behaviour was observed for all five seeds, which indicates that the policy indeed overfits to the NUS. For the policies trained with the ABUS this was not observed. This could indicate that the policy can learn to exploit some of the shortcomings of the trained NUS.

6.2 Human Evaluation

The results of the human evaluation are shown in Table 4 for 250 dialogues per policy. In Table 4 policies are marked using an ID ($\mathcal{U}_\alpha$) that translates to results in Tables 2 and 3. 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.

Training Simulator	Human Evaluation	
	Rew.	Suc.
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 4: 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 2 and 3 since all user goals were achievable.

7 Conclusion

We introduced the Neural User Simulator (NUS), which uses the system's response in its semantic form as input and gives a natural language response. It thus needs *less labelling* of the training data than User Simulators that generate a response in semantic form. 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 the Agenda-Based User Simulator 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.

Acknowledgements

This research was partly funded by the EPSRC grant EP/M018946/1 Open Domain Statistical Spoken Dialogue Systems. Florian Kreyssig is supported by the Studienstiftung des Deutschen Volkes. Paweł Budzianowski is supported by the EPSRC and Toshiba Research Europe Ltd.

References

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.

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*.

Iñigo Casanueva, Paweł Budzianowski, Pei-Hao Su, Stefan Ultes, Lina Rojas-Barahona, Bo-Hsiang Tseng, and Milica Gašić. 2018. Feudal reinforcement learning for dialogue management in large domains. In *Proc. NAACL 2018*.

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*.

Cheng Chang, Runzhe Yang, Lu Chen, Xiang Zhou, and Kai Yu. 2017. Affordable on-line dialogue policy learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2200–2209.

Lu Chen, Xiang Zhou, Cheng Chang, Runzhe Yang, and Kai Yu. 2017. Agent-aware dropout dqn for safe and efficient on-line dialogue policy learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2454–2464.

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.

Mehdi Fatemi, Layla El Asri, Hannes Schulz, Jing He, and Kaheer Suleman. 2016. Policy networks with two-stage training for dialogue systems. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 101–110.

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.

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*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization.

Sangjun Koo, Seonghan Ryu, and Gary Geunbae Lee. 2015. Implementation of generic positive-negative tracker in extensible dialog system. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 798–805. IEEE.

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.

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, 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 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.

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.

Pararth Shah, Dilek Hakkani-Tür, Gokhan Tür, Abhinav Rastogi, Ankur Bapna, Neha Nayak, and Larry Heck. 2018. Building a conversational agent overnight with dialogue self-play. *arXiv preprint arXiv:1801.04871*.

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.

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.

Gellért Weisz, Paweł Budzianowski, Pei-Hao Su, and Milica Gašić. 2018. Sample efficient deep reinforcement learning for dialogue systems with large action spaces. *arXiv preprint arXiv:1802.03753*.

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.