Abstract
Dialogue policy optimisation via reinforcement learning requires a large number of training interactions, which makes learning with real users time consuming and expensive. Many set-ups therefore rely on a user simulator instead of humans. These user simulators have their own problems. While hand-coded, rule-based user simulators have been shown to be sufficient in small, simple domains, for complex domains the number of rules quickly becomes intractable. State-of-the-art data-driven user simulators, on the other hand, are still domain-dependent. This means that adaptation to each new domain requires redesigning and retraining. In this work, we propose a domain-independent transformer-based user simulator (TUS). The structure of our TUS is not tied to a specific domain, enabling domain generalisation and learning of cross-domain user behaviour from data. We compare TUS with the state of the art using automatic as well as human evaluations. TUS can compete with rule-based user simulators on pre-defined domains and is able to generalise to unseen domains in a zero-shot fashion.

1 Introduction
Task-oriented dialogue systems are designed to help users accomplish specific goals within a particular task such as hotel booking or finding a flight. Solving this problem typically requires tracking and planning (Young, 2002). In tracking, the system keeps track of information about the user goal from the beginning of the dialogue until the current dialogue turn. In planning, the dialogue policy makes decisions at each turn to maximise future rewards at the end of the dialogue (Levin and Pieraccini, 1997). The system typically needs thousands of interactions to train a usable policy (Schatzmann et al., 2007; Pietquin et al., 2011; Li et al., 2016; Shi et al., 2019). The amount of interactions required makes learning from real users time-consuming and costly. It is therefore appealing to automatically generate a large number of dialogues with a user simulator (US) (Eckert et al., 1997).

Rule-based USs are interpretable and have shown success when applied in small, simple domains. However, expert knowledge is required to design their rules and the number of rules needed for complex domains quickly becomes intractable (Schatzmann et al., 2007). In addition, handcrafted rules are unable to capture human behaviour to its fullest extent, leading to sub-optimal performance when interacting with real users (Schatzmann et al., 2006).

Data-driven USs on the other hand can learn user behaviour directly from a corpus. However, they are still domain-dependent. This means that in order to accommodate an unseen domain one needs to collect and annotate a new dataset, and retrain or even re-engineer the simulator.

We propose a transformer-based domain-independent user simulator (TUS). Unlike existing data-driven simulators, we design the feature representation to be domain-independent, allowing the simulator to easily generalise to new domains without modifying or retraining the model. We utilise a transformer architecture (Vaswani et al., 2017) so that the input sequence can have a variable length and dynamic order. The dynamic order takes into account the user’s priorities and the varying input length enables the US to incorporate system actions in a seamless manner. TUS predicts the value of each slot and the domains of the current turn, allowing the model to optimise its performance in multiple granularities. By disentangling the user behaviour from the domains, TUS can learn a more general user policy to train the dialogue policy.
We compare policies trained with our TUS to policies trained with other USs through indirect and direct evaluation as well as human evaluation. The results show that policies trained with TUS outperform those that are trained with another data-driven US and are on par with policies trained with the agenda-based US (ABUS). Moreover, the policy generalises better when evaluated with a different US. Automatic and human evaluations on our zero-shot study show that leave-one-domain-out TUS is able to generalise to unseen domains while maintaining a comparable performance to ABUS and TUS trained on the full training data.

2 Related Work

The quality of a US has a significant impact on the performance of a reinforcement-learning based task-oriented dialogue system (Schatzmann et al., 2005). One of the early models include an N-gram user simulator proposed by Eckert et al. (1997). It uses a 2-gram model $P(a_u | a_m)$ to predict the user action $a_u$ according to the system action $a_m$. Since it only has access to the latest system action, its behaviour can be illogical if the goal changes. Therefore, models which can take into account a given user goal were introduced (Georgila et al., 2006; Eshky et al., 2012). The Bayesian model of Daubigney et al. (2012) predicts the user action based on the user goal, and hidden Markov models are used to model the user and the system behaviour (Cuayáhuitl et al., 2005). The graph-based US of Scheffler and Young (2002) combines all possible dialogue paths in a graph. It can generate reasonable and consistent behaviour, but is impractical to implement, since extensive domain knowledge is required.

The agenda-based user simulator (ABUS) (Schatzmann et al., 2007) models the user state as a stack-like agenda, ordered according to the priority of the user actions. The probabilities of updating the agenda and choosing user actions are set manually or learned from data (Keizer et al., 2010). Still, the stacking and popping rules are domain-dependent and need to be designed carefully.

To build a data-driven model, the sequence-to-sequence (Seq2Seq) model structure is widely used. El Asri et al. (2016) propose a Seq2Seq semantic level US with an encoder-decoder structure. Each turn is fed into the encoder recurrent neural network (RNN) and embedded as a context vector. Then this context vector is passed to the decoder RNN to generate user actions. To add new domains, it is necessary to modify the domain-dependent feature representation and retrain the model.

Instead of generating semantic level output, the neural user simulator (NUS) by Kreyssig et al. (2018) generates responses in natural language, thus requiring less labeling, at the expense of interpretability. However, its feature representation is still domain-dependent.

A variational hierarchical Seq2Seq user simulator (VHUS) is proposed by Gür et al. (2018). Instead of designing dialogue history features, the model encodes the user goal and system actions with a vector using an RNN, which alleviates the need of heavy feature engineering. However, the inputs are represented as one-hot encodings, which are also dependent on the ontology. In addition, the output generator is not constrained by the ontology in any way, so it can generate impossible actions.

As shown in Fig. 1, ABUS and graph-based models are domain-dependent and require significant design efforts. Data-driven models such as Seq2Seq, NUS, and VHUS can learn from data, but are constrained by the underlying domain. NUS generates natural language responses, which requires less labeling, but comes with reduced interpretability.

Shi et al. (2019) compared different ways to build a US and indicated that the data-driven models suffer from bias in the corpus. If some actions are rare in the corpus, the model cannot capture them. Thus, the dialogue policy cannot explore all possible paths during training with the data-driven USs. It is important to learn more general human behaviour to reduce the impact of the corpus bias.
3 Problem Description

Task-oriented dialogue systems are defined by a given ontology, which specifies the concepts that the system can handle. The ontology can include multiple domains. In each domain, there are informable slots, which are the attributes that users can assign values to, and requestable slots, which are the attributes that users can query. For example, in Fig. 2 the user goal has two domains, “hotel” and “restaurant”. The slot Area is an informable slot with the value North in domain “hotel” and Addr is a requestable slot in domain “restaurant”. The system state records the slots and values mentioned in the dialogue history. A US for task-oriented dialogue systems needs to provide coherent responses according to a given user goal \( G = \{ \text{domain}_1 : [(\text{slot}_1, \text{value}_1), (\text{slot}_2, \text{value}_2), \ldots], \ldots \} \). The domains, slots and values are selected from the ontology.

The user action is composed of user intents, domains, slots, and values. We consider user intents that appear in the MultiWOZ dataset (Budzianowski et al., 2018). It is of course possible to consider arbitrary intents within the same model architecture, as long as they are defined a priori\(^2\). The two possible user intents we consider are Inform and Request. With Inform, the user can provide information, correct the system or confirm the system’s recommendations. When a user goal cannot be fulfilled, the user can also randomly select a value from the ontology and change the goal. With Request, the user can request information about certain slots.

The system action is similar to the user action, but there exist more (system) intents. For example, the system can provide suggestions to users with the intent Recommendation and make reservations for users with the intent Book. More system intents can be found in Appendix A.

We view user simulation in a task-oriented dialogue as a sequence-to-sequence problem. For each turn \( t \), we extract the input feature vectors \( V^t = [v_1^t, v_2^t, \ldots, v_n^t] \) is generated based on the system actions and the user goal, where \( v_i^t \) is the feature vector of slot \( s_i \) and \( n_t \) is the length of the input list in turn \( t \), \( V^t \). We explain the feature representation in detail in Sec. 4.1. Inspired by ABUS, which models the user state as a stack-like agenda, the length of input list \( n_t \) at each turn \( t \) varies by taking into account slots mentioned in the system’s action. For example, in Fig. 3 the input list \( V^0 \) only contains the slots in the user goal at the first turn. Then the system mentions a slot not in the user goal, Hotel-Price. So in turn 1 the length of input list \( V^1 \) is \( n_1 = n_0 + 1 \) because one slot is inserted into the input list \( V^1 \). The whole input sequence to the model is \( V_{\text{input}} = [v_{\text{CLS}}, v_1^t, v_2^t, \ldots, v_{\text{SEP}}, v_{t-1}^t, \ldots, v_{\text{SEP}}] \), where \( v_{\text{CLS}} \) is the representation of [CLS] and \( v_{\text{SEP}} \) is the representation of [SEP].

The user policy network is a transformer (Vaswani et al., 2017; Devlin et al., 2019). We choose this structure because transformers are able to handle input sequences of arbitrary lengths and to capture the relationship between slots thanks

\(^2\)We note that intents are not normally dependent on the domain but rather on the kind of dialogue that is being modeled, e.g. task-oriented or chit-chat.
4.1 Domain-independent Input Features

We design the input feature representation $v^t_i$ of each slot $s_i$ in turn $t$ consisting of a set of sub-vectors, all of which are domain-independent. For better readability, we drop the slot index $i$ and the turn index $t$, i.e., we write $v$ for $v^t_i$.

4.1.1 Basic Information Features

Inspired by the feature representation proposed in El Asri et al. (2016), we use a feature vector $v_{\text{basic}}$ that is composed of binary sub-vectors to represent the basic information for each slot. Each slot has two value vectors: $v_{\text{value}}^{\text{sys}}$ represents the value in the system state, and $v_{\text{value}}^{\text{user}}$ represents the value in the user goal. Each value vector is a 4-dimensional one-hot vector, with coordinates encoding “none”, “?” “don’t care” or “other values”, in this order. For example, in turn 1 in Fig. 2, for slot Hotel-Price, $v_{\text{value}}^{\text{sys}} = [1, 0, 0, 0]$, i.e., “none”, because it is not in the user goal, and $v_{\text{value}}^{\text{user}} = [0, 1, 0, 0]$, i.e., “?” because the system requests it.

The slot type vector $v_{\text{type}}$ is a 2-dimensional vector which represents whether a slot is in the user goal as a constraint or a request. For example,
in Fig. 2 for Hotel-Area $v_{type} = [1, 0]$ (constraint), while for Hotel-Name $v_{type} = [0, 1]$ (request). A value of $[0, 0]$ means that the slot is not included in the user goal.

The state vector $v_{ful}$ encodes whether or not a constraint or informative slot has been fulfilled. The value is set to 1 if the constraint has been fulfilled, and to 0 otherwise. The vector $v_{first}$ similarly encodes whether a slot is mentioned for the first time.

The basic information feature vector $v_{basic}$ is the concatenation of these vectors, i.e.,

$$v_{basic} = v_{user} \oplus v_{sys} \oplus v_{type} \oplus v_{ful} \oplus v_{first} \quad (2)$$

### 4.1.2 System Action Features

The system action feature vector $v_{action}^{system}$ encodes system actions in each turn. There are two kinds of system actions, general actions and domain-specific actions. The general actions are composed only with general intents, such as “reqmore” and “bye”. For example, general-reqmore(). The feature vector of general actions $v_{gen}$ is a multi-hot encoding of whether or not a general intent appears in the dialogue. With a total number of $n_{gen}$ general intents, for each $k \in \{1, \ldots, n_{gen}\}$, the $k$-th entry of $v_{gen}$ is set to 1 if the $k$-th general intent is part of the system act.

On the other hand, domain-specific actions are composed with domains, slots, values, and domain-specific intents such as “recommend” and “select”. For example, Recom(Hotel-Area=South). Each domain-specific action vector $v_{spec}$, with the domain-specific $j$-th intent, $j \in \{1, \ldots, n_{spec}\}$, where $n_{spec}$ is the total number of domain-specific intents, is represented by a 3-dimensional one-hot encoding that describes whether the value is “none”, “?” or “other values”.

The final action representation $v_{action}^{system}$ is formed by concatenating $n_{spec}$ domain-specific action representations together with the general action representation, i.e.,

$$v_{action}^{system} = v_{spec0} \oplus \cdots \oplus v_{spec_{n_{spec}}} \oplus v_{gen}. \quad (3)$$

For the slot Hotel-Area in Fig. 3, we have a vector for each intent. For the intent “recommend” $v_{spec0} = [0, 0, 1]$, which means that “other values” (in this case South) are mentioned. For all other domain-specific intents, the vectors are $[1, 0, 0]$ since no value is mentioned. In terms of the general intents, only “reqmore” is mentioned, so $v_{gen}[1] = 1$, as “reqmore” is the first general intent.

### 4.1.3 User Action Features

The output vector from the previous turn $O^{t-1}$ is also included in the input features of the next turn $t$ to take into account what has been mentioned by the US itself, i.e. for slot $s_i$ in turn $t$, the user action feature $v_{action}^{user} = O^{t-1}$.

### 4.1.4 Domain and Slot Index Features

In some cases, multiple slots may share the same basic feature $v_{basic}$, system action feature $v_{action}^{system}$ and user action feature $v_{action}^{user}$. This similarity in features of different slots makes it difficult for the model to distinguish one slot from another, despite the positional encoding. In particular, it is challenging for the model to learn the relationship between turns for a given slot because the number and the order of slots vary from one turn to the next. This may lead to over-generation: the model selects all slots with the same feature vector.

To counteract this issue, we introduce the index feature $v_{index}$, which consists of the domain index feature $v_{index}^{domain} \in \{0, 1\}^{l_d}$ and the slot index feature $v_{index}^{slot} \in \{0, 1\}^{l_s}$, where $l_d$ is the maximum number of domains in a user goal and $l_s$ is the maximum number of slots in any given domain.

To make the index feature ontology-independent, for a particular slot, $v_{index}$ remains consistent throughout a dialogue, but varies between dialogues. The order of the index in each dialogue is determined by the order in the user goal. For example, the “hotel” domain can be the first domain in one user goal of the first dialogue, and the second domain in the next.

Then for each slot in each turn the input feature vector $v$ is formed by concatenating all sub-vectors:

$$v = v_{basic} \oplus v_{action}^{system} \oplus v_{action}^{user} \oplus v_{index}. \quad (4)$$

An example of $v$ for slot Hotel-Area is shown in Fig. 3 based on the dialogue history in Fig. 2. Examples of how the feature representation is constructed can be seen in Appendix D.

### 4.2 Domain Prediction

Inspired by solving downstream tasks using BERT (Devlin et al., 2019), we utilise the output of $[CLS]$, $o_{CLS}$, to predict which domains are considered in turn $t$. A multi-label classification...
problem. The domain loss $loss_{domain}$ measures the difference between the output $o_{CLS}$ and the target $y_{CLS}$ for each turn by binary cross entropy (BCE). The final loss function is defined as

$$loss = loss_{slots} + loss_{domain}. \quad (5)$$

5 Experimental Setup

5.1 Supervised Training for TUS

Our model is implemented in PyTorch (Paszke et al., 2019) and optimised using the Adam optimiser (Kingma and Ba, 2015) with learning rate $5 \times 10^{-4}$. The dimension of the input linear layer is 100, the number of the transformer layers is 2, and the dimension of the output linear layer is 6. The maximum number of domains $l_d$ is 6 and the maximum number of slots in one domain $l_s$ is 10. During training, the dropout rate is 0.1.

We train our model\footnote{https://gitlab.cs.uni-duesseldorf.de/general/dsml/tus_public} on the MultiWOZ 2.1 dataset (Eric et al., 2020), consisting of dialogues between two humans, one posing as a user and the other as an operator. The dialogues in the dataset are complex because there may be more than one domain involved in one dialogue, even in the same turn. During training and testing with the dataset, the order of slots in the input list is derived from the data, which means slot $s_i$ is before slot $s_{i+1}$ if the user mentioned slot $s_i$ first. For inference without the dataset, such as when using TUS to train a dialogue policy, the order of slots is randomly generated.

We measure how well a US can fit the dataset by precision, recall, F1 score, and turn accuracy. The turn accuracy measures how many model predictions per turn are identical to the corpus, based on the oracle dialogue history.

5.2 Training Policies with USs

User simulators are designed to train dialogue systems, thus a better user simulator should result in a better dialogue system. We train different dialogue policies by proximal policy optimization (PPO) (Schulman et al., 2017), a simple and stable reinforcement learning algorithm, with ABUS, VHUS, and TUS as USs in the ConvLab-2 framework (Zhu et al., 2020). The policies are trained for 200 epochs, each of which consists of 1000 dialogues. The reward function gives a reward of 80 for a successful dialogue and of -1 for each dialogue turn, with the maximum number of dialogue turns set to 40. For failed dialogues, an additional penalty is set to -40. Each dialogue policy is trained on 5 random seeds. The dialogue policies are then evaluated using all USs by cross-model evaluation (Schatzmann et al., 2005) to demonstrate the generalisation ability of the policy trained with a particular US when evaluated with a different US.

5.3 Leave-one-domain-out Training

To evaluate the ability of TUS in handling unseen domains, we remove one domain during supervised learning of TUS. The leave-one-domain-out TUSs are used to train dialogue policies with all possible domains. For example, TUS-noHotel is trained on the dataset without the “hotel” domain. During policy training, the user goal is generated randomly from all possible domains.

Some domains in MultiWOZ may share the same slots, such as “restaurant” and “hotel” domains which contain property-related slots, e.g. “area,” “name,” and “price range.” However, the corpus also includes domains that are quite different from the rest. For example, the “train” domain which contains many time-related slots such as “arrival time” or “departure time”, as well as unique slots such as “price” and “duration.” The different properties of the domains will allow us to study the zero-shot transfer capability of the model.

5.4 Human Evaluation

Following the setting in Kreyssig et al. (2018), we select 2 of the 5 trained versions of each dialogue policy for evaluation in a human trial: the version performing best on ABUS, and the version performing best in interaction with TUS. The results of the two versions are averaged. For each version we collect 200 dialogues, which means there are 400 dialogues for each policy in total. Dialogue policies trained with VHUS significantly underperform, so we only consider policies trained with ABUS or TUS for the human trial (see Table 1). The best and the worst policies in the leave-one-domain-out experiment are also included to see the upper and lower bound of the zero-shot domain generalisation performance.

Human evaluation is performed via DialCrowd (Lee et al., 2018) connected to Amazon Mechanical Turk\footnote{https://www.mturk.com/}. Users are provided with a randomly generated user goal and are required to interact with our systems in natural language.
Table 1: The success rates of policies trained on ABUS, VHUS, and TUS when tested on various USs.

<table>
<thead>
<tr>
<th>US for training</th>
<th>US for evaluation</th>
<th>ABUS</th>
<th>VHUS</th>
<th>TUS</th>
<th>avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABUS</td>
<td>0.93</td>
<td>0.09</td>
<td>0.58</td>
<td>0.53</td>
<td>0.54</td>
</tr>
<tr>
<td>VHUS</td>
<td>0.62</td>
<td>0.11</td>
<td>0.37</td>
<td>0.36</td>
<td>0.38</td>
</tr>
<tr>
<td>TUS</td>
<td>0.79</td>
<td>0.10</td>
<td>0.69</td>
<td>0.53</td>
<td>0.56</td>
</tr>
</tbody>
</table>

6 Experimental Results

6.1 Cross-model Evaluation

The results of our experiments are shown in Table 1. The policy trained with TUS performs well when evaluated with ABUS, with 10% absolute improvement in the success rate over its performance on TUS. On the other hand, while a policy trained with ABUS performs almost perfectly when evaluated with ABUS, the performance drops significantly, by 35% absolute, when this policy interacts with TUS. This signals that, in the case of ABUS, the policy overfits to the US used for training, and is not able to generalise well to the behaviour of other USs. We found that VHUS is neither able to train nor to evaluate a multi-domain policy adequately. This was also observed in the experiments by Takanobu et al. (2019). We suspect that this is due to the fact that VHUS was designed to operate on a single domain and does not generalise well to the multi-domain scenario. To the best of our knowledge, no other data-driven US has been developed for the multi-domain scenario.

The success rates of policies trained with ABUS and TUS during training, evaluated with both US, are shown in Fig. 4. Each of the systems is trained 5 times on different random seeds. We report the average success rate as well as the standard deviation. Although the policy trained with TUS is more unstable when evaluated on ABUS, it still shows an improvement from the initial policy, converging at around 79%. On the other hand, the policy trained with ABUS and evaluated with TUS barely show any improvements.

6.2 Impact of features and loss functions

We conduct an ablation study to investigate the usefulness of the proposed features and loss functions. The result is shown in Table 2. First, we measure the performance of the basic model which uses only the basic information feature $v_{\text{basic}}$, the system action feature $v_{\text{sys}}$, and the user action feature $v_{\text{user}}$ as the input. While this model can have a high recall rate, the precision and the turn accuracy are fairly low. We deduce that without the index features the model cannot distinguish the difference between slots and therefore tends to select slots of the same slot type in one turn. For example, it provides all constraints in the first turn, which leads to high recall and over-generation.

Analysis of the generated user actions shows that the basic model tends to mention four or more slots in the first turn. This is unnatural, since human users tend to only mention one or two slots at the beginning of a dialogue. More details about the average slots per turn can be found in Appendix B.

After adding the index feature $v_{\text{index}}$, the recall rate is decreased by 17% absolute, but the turn accuracy is increased by 35% absolute, along with improvements on the precision and the F1 score. Furthermore, the average number of slots per turn is closer to that of a real user. Although the recall rate with respect to the target in the data is decreased, this is not necessarily a concern since in dialogue there are many different plausible actions for a given context. For example, when searching for a restaurant, we may provide the information of the area first, or the food type. The order of
Table 3: The success rates of dialogue policies trained with leave-one-domain-out TUSs. For example, the TUS-noAttr model is trained without the “attraction” domain. The sum of all removed data is more than 100% because some dialogues have multiple domains. We report results on all domains.

6.3 Zero-shot Transfer

We test the capability of the model to handle unseen domains in a zero-shot experiment. In a leave-one-domain-out fashion we remove dialogues involving one particular domain when training the US. The share of each domain in the total dialogue data ranges from 19.60% to 45.21%. During dialogue policy training we sample the user goal from all domains. As presented in Table 3, removing one domain from the training data when training the US does not dramatically influence the policy on the corresponding domain. The final performance of the policies trained with leave-one-domain-out TUSs is still reasonably comparable to the policy trained with the full TUS. This is especially noteworthy considering the substantial amount of data removed during US training and the difference between each domain.

We observe that the model is able to learn about the removed domain from the other domains, although the removed domain is different from the remaining ones. For example, the “train” domain is very different from “attraction”, “restaurant”, and “hotel”, and it is more complex than “taxi”, but TUS-noTrain still performs reasonably well on the “train” domain. This signals that the model can do zero-shot transfer by leveraging other domain information. The worst performance on the “train” domain happens instead when the “hotel” domain is removed, i.e. the domain with the most substantial amount of data.

Our results also show that some domains are more sensitive to data removal than others, irrespective of which domain is removed. This indicates that some domains are more involved and simply require more training data. This result demonstrates that TUS has the capability to handle new unseen domains without modifying the feature representation or retraining the model. It also shows that our model is sample-efficient.

6.4 Human Evaluation

The result of the human evaluation is shown in Table 4. In total, 156 users participated in the human evaluation. The number of interactions per user ranges from 10 to 80. The success rate measures whether the given goal is fulfilled by the system and the overall rating grades the system’s performance from 1 star (poor) to 5 stars (excellent). TUS is able to achieve a comparable success rate as ABUS, without domain-specific information, and even scores slightly better in terms of overall rating. We were not able to observe any statistically significant differences between ABUS and TUS in the human evaluation. For leave-one-domain-out mod-
els, the performance of TUS-noAttr is similar to that one of ABUS and TUS without a statistically significant difference. We do however observe a statistically significant decrease in the success rate of TUS-noHotel when compared to TUS and ABUS ($p < 0.05$). This is unsurprising as the hotel domain accounts for 40.15% of the training data. For both TUS-noAttr and TUS-noHotel, the success rate on the domain “attraction” is comparable to TUS and ABUS, but the success rate on the domain “hotel” is relatively low. As observed in the simulation, removing a domain does not decrease the success rate in the corresponding domain as the feature representation is domain agnostic. Instead, it impacts domains which need plenty of data to learn.

7 Conclusion

We propose a domain-independent user simulator with transformers, TUS. We design ontology-independent input and output feature representations. TUS outperforms the data-driven VHUS and it has a comparable performance to the rule-based ABUS in cross-model evaluation. Human evaluation confirms that TUS can compete with ABUS even though ABUS is based on carefully designed domain-dependent rules. Our ablation study shows that the proposed features and loss functions are essential to model natural user behavior from data. Lastly, our zero-shot study shows that TUS can handle new domains without feature modification or model retraining, even with substantially fewer training samples.

In future work, we would like to learn the order of slots and add output language generation to make the behaviour of TUS more human-like. Applying reinforcement learning to this model would also be of interest.

Acknowledgments

We would like to thank Ting-Rui Chiang and Dr. Maxine Eskenazi from Carnegie Mellon University for their help with the human trial. This work is a part of DYM0 project which has received funding from the European Research Council (ERC) provided under the Horizon 2020 research and innovation programme (Grant agreement No. STG2018 804636). N. Lubis, C. van Niekerk, M. Heck and S. Feng are funded by an Alexander von Humboldt Sofja Kovalevskaja Award endowed by the German Federal Ministry of Education and Research. Computational infrastructure and support were provided by the Centre for Information and Media Technology at Heinrich Heine University Düsseldorf. Computing resources were provided by Google Cloud.

References


Aciel Eshky, Ben Allison, and Mark Steedman. 2012. Generative goal-driven user simulation for dialog


All System Intents

<table>
<thead>
<tr>
<th>type</th>
<th>intents</th>
</tr>
</thead>
<tbody>
<tr>
<td>general</td>
<td>welcome, reqmore, bye, thank, greet</td>
</tr>
<tr>
<td>domain-</td>
<td>recommend, inform, request, select,</td>
</tr>
<tr>
<td>specific</td>
<td>book, nobook, offerbook, offerbooked,</td>
</tr>
<tr>
<td></td>
<td>nooffer</td>
</tr>
</tbody>
</table>

Table 5: All system intents in the MultiWOZ 2.1 dataset

Average Action Length in Each Turn

The average number of slots mentioned by TUS in each turn when interacting with the rule-based dialogue system is shown in Fig. 5. When the index feature $v_{index}$ and the domain loss $loss_{domain}$ are added, TUS can deal with the over-generation problem and behave more similarly to what is observed in the corpus.

Success Rates of Leave-one-domain-out Training

The training success rates of dialogue policies trained with leave-one-domain-out TUSs, which are evaluated on TUS, are shown in Fig. 6. In comparison to the full TUS, the leave-one-domain-out TUSs are more unstable, but they can achieve a comparable success rate at the end.

An example for the input feature representation

The list of input feature vectors and output sequence are presented on Fig. 7 based on Fig. 2.

For turn 0, $V^0$ only includes 4 vectors from the user goal. For turn 1, the system mentions slot Hotel-Price, which is not in the user goal, so the feature vector of slot Hotel-Price is inserted into $V^1$, where the 1-st dimension of $v_{slot}$ is 1 because domain Hotel is the first domain in this conversation and the 3-rd dimension of $v_{index}$ is 1 because it is the third slot in domain Hotel.

In comparison between the feature vectors of slot Hotel-Area in turn 0, $v_{1}^0$, and turn 1, $v_{1}^1$, the $v_{sys}$ and $v_{spec}$ are different because of the system’s domain-specific action Recom(Hotel-Area=South). The system also mentioned a general action, general-reqmore(), thus $v_{gen}$ is changed.
Figure 7: The input and feature representation according to Fig. 2. $v_{CLS}$ and $v_{sep}$ are ignored in this graph.

addition, this slot is first mentioned at turn 0, so $v_{\text{first}}$ is changed from 0 to 1. Similarly, $v_{\text{action}}$ is also modified according to the user action. On the other hand, $v_{\text{value}}$ is the same because the user does not update its goal, $v_{\text{type}}$ is not changed because the slot is still a constraint, and $v_{\text{ful}}$ is 0 because it has not been fulfilled yet. $v_{\text{domain}}$ and $v_{\text{slot}}$ are also the same through the whole conversation.

E Example Dialogue Generated by TUS

An example dialogue with a multi-domain user goal is shown in Fig. 8. It shows that TUS is able to switch between different domains (from turn 2 to 6), respond to the system’s requests, and generate multi-domain actions (in turn 5).

Turn 0
USR: Inform(Hotel-Area=north, Hotel-Stars=0, Hotel-Parking=yes)
SYS: Inform(Hotel-Parking=free)

Turn 1
USR: Request(Hotel-Type)
SYS: Inform(Hotel-Type=guesthouse)

Turn 2
USR: Inform(Hotel-Stars=0, Hotel-Type=guesthouse, Hotel-Area=north, Hotel-Parking=free, Hotel-Parking=north)
SYS: Inform(Hotel-Area=north)

Turn 3
USR: Inform(Attr-Type=college)
SYS: Inform(Attr-Type=college, Attr-Hotel=free)

Turn 4
USR: Request(Taxi-Phone, Taxi-Car)
SYS: Request(Taxi-Car)

Turn 5
USR: Inform(Taxi-Car=audi, Taxi-Phone=6125228555, Taxi-Price=free)
SYS: Inform(Taxi-Car=audi, Taxi-Phone=6125228555, Taxi-Price=free)

Turn 6
USR: Request(Attr-Post)
SYS: Inform(Attr-Post=cb23bu)

Turn 7
USR: general-bye()
SYS: general-bye()