

Graph Neural Net-based User Simulator

Xinrui Nie, Zehao Lin, Xinjing Huang, and Yin Zhang**

Zhejiang University, Hangzhou 310007, China

{3150104023, georgelin, huangxinjing, zhangyin98}@zju.edu.cn

<http://www.dcd.zju.edu.cn>

Abstract. User Simulators are major tools that enable offline training of task-oriented dialogue systems. To efficiently utilize semantic dialog data and generate natural language utterances, user simulators based on neural network architectures are proposed. However, existing neural user simulators still rely on hand-crafted rules, which is difficult to ensure the effectiveness of feature extraction. This paper proposes the Graph Neural Net-based User Simulator (GUS), which constructs semantic graphs from the corpus and uses them to build Graph Convolutional Network (GCN) to extract feature vectors. We tested our model on examined public dataset and also made conversation with real human directly to verify the effectiveness. Experimental results show GUS significantly outperforms several state-of-the-art user simulators.

Keywords: User Simulator · Graph Neural Networks · Dialogue Systems.

1 Introduction

The Dialogue Manager of Spoken Dialogue System (SDS) is generally trained by reinforcement learning [2]. Through the control of the reward function, the dialog manager learns optimal policy by maximizing the cumulative rewards. The process of selecting the next action is called the policy. To train a well-functioning policy, a large amount of dialog data is needed. But considering the low effectiveness and high cost, it is unrealistic to obtain large-scale dataset from interactions with real users. Either we can't directly learn policy from the corpus since it is static and the accessed dialog state space is very limited. Therefore, user simulator came into being. User simulator is trained from the corpus to learn how real users will respond in a given context. It transforms a static corpus into a dynamic tool that can generate any number of dialogues and explore the unrecorded state space.

In recent years, user simulators have evolved from hand-crafted rules-based models to deep learning models. Works [1, 13, 14] try to introduce neural network and expect a better performance with less human efforts than rule-based models. However, these user simulators still use hand-crafted rules and thus suffers from migrating to other domains. To relieve this problem, we adopt Graph Neural

** Corresponding author

Network (GNN) to automatically learn to represent not only utterance sequence but also semantic structure as dialogue state.

In this paper, we propose the Graph Neural Net-based User Simulator (GUS), which takes the semantic data as inputs and generates natural language. Its core consists of a graph generator, a GCN-based feature extractor, and a sequence-to-sequence model. The graph generator constructs the semantic graphs from the semantic data in the corpus by our designed mechanism. The GCN feature extractor extracts feature from the dialogue history by applying convolution on the semantic graphs. The sequence-to-sequence model contains an RNN encoder, which encodes the feature vector sequence, and an RNN decoder, which outputs natural language. In addition, GUS can generate its own goals and change goals during the dialogue. This helps to train a more complex dialog manager policy.

GUS is trained on DSTC2 (Dialog State Tracking Challenge 2) dataset, which is built on the restaurant recommendation domain. Its ontology defines: (1) informable slots, which are user-selectable restaurant attributes (such as food, area, price range) (2) requestable slots, which are attributes that users can request (such as address, phone number, zip code) (3) all restaurant names. Each attribute of the restaurant appears as a slot-value pair (e.g. `area=north`).

We use BLEU to measure the performance of GUS and visually verify its validity by establishing a real dialogue history and observing the model outputs. We compare the performance of GUS with NUS, Sequence-to-Sequence User Simulator (S2SUS) and Graph Convolutional Encoded User Simulator (GCEUS). In the two evaluation tasks, GUS is the best in all models.

2 Related Work

User simulator can be classified according to the abstraction level of modeling dialogues [4], which is roughly divided into semantic level and word level. Up to now, most user simulators work on the semantic level, which needs to handle user dialog acts and corresponding slot-value pairs. The first user simulator [5] proposed a simple bi-gram model to predict the user action a_u based on the system action a_s as $p = P(a_u|a_s)$. The advantage lies in pure probability modeling and complete domain independence. However, it can't produce consistent user behavior, since it only focuses on the last action of the system and the full dialogue history is not modeled. This leads to user utterances often being meaningless in a longer context. Scheffler [6] tried to solve the problem of inconsistency by introducing a fixed goal structure. But it requires a large amount of domain-specific knowledge. Pietquin [7] combined the characteristics of the former two models. The core idea is to condition the probability to explicit user goals: $P(\text{provide } A_x | \text{request } A_x, \text{goal})$ (A_x represents an attribute). This allows it to explicitly model the dependencies between user acts and goals. Georgila [8] used the Markov model to describe the user's current state with a large feature vector. However, the model is not conditioned on user goals, which makes it impossible to know whether the goal has been completed, and therefore cannot be used to train the dialogue manager's policy. Chandramohan [9] regarded

the user simulation as an inverse reinforcement learning problem, and modeled users as a decision-making agent. But the model does not incorporate user goals either. The most prominent user simulator is ABUS [10] (Agenda-based User Simulator), which models state transition and user action generation through a stack structure called agenda. Based on hand-crafted rules, ABUS explicitly encodes user goals and dialogue history, and has a corresponding update strategy when the system is unable to complete the goal.

3 Proposed Framework

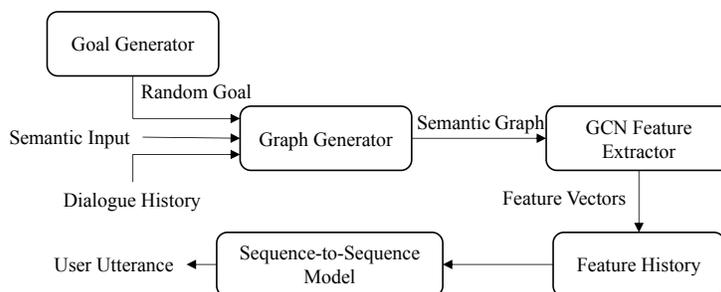


Fig. 1. General Architecture of the Graph Neural Net-based User Simulator (GUS)

The above user simulators are all built on the semantic level. Many problems arise from this approach. First, research [11] shows that the Spoken Language Understanding and Belief Tracking module in SDS should be jointly trained as a single entity, which is impossible when user simulator does not output natural language. Second, all user sentences must be correctly semantically labeled, which is costly. Third, user simulator is sometimes incremented with an error model which reproduces the speech recognizer errors. SDS performs better when the error model is more consistent with the characteristics of the speech recognizer [12]. Since speech recognition errors are word-level, the error model based on semantic modeling may be worse than the error model based on natural language modeling.

The study of word-level user simulators has begun to reheat in recent years. Crook [13] used the sequence-to-sequence model to train word-level user simulators, but did not incorporate user goals, so it could not be used for policy optimization. Li [14] added a natural language generation module to ABUS to convert it into a word-level user simulator. Kreyssig [1] proposed the NUS architecture, which takes the semantic input of SDS and generates feature vectors through hand-crafted rules. Then natural language is generated through the sequence-to-sequence model.

The overall architecture of GUS is shown in Fig. 1. At the beginning of the dialogue, an initial random goal G_0 is generated by the goal generator. In dialogue turn T , the semantic input of the system statement, the current user goal G_T , and the dialogue history are fed into the graph generator to generate a series of semantic graphs: $g_{1:T} = g_1 \dots g_T$, which are then fed into the GCN feature extractor and used to generate a sequence of feature vectors $\mathbf{v}_{1:T} = \mathbf{v}_1 \dots \mathbf{v}_T$. Note these vectors can be viewed as a summary of each turn, thus dialogue history can be modeled by applying sequence-to-sequence model on the vector sequence to generate a user utterance. $\mathbf{u}_T = \omega_0 \dots \omega_{n_T}$ with length n_T . Values corresponding to all slots that appear in the user utterances are replaced with tokens, a process called delexicalisation. When the system statement indicates that no venue matches the current user goal, the goal generator randomly modifies the goal.

3.1 Goal Generator

At the beginning of the dialogue, the goal generator generates an initial random goal $G_0 = C_0$, which represents a set of constraints imposed by the user on the final venue, such as (`food=indian`, `pricerange=cheap`). The probability of occurrence of each informable slot in C_0 is defined by the ontology. In DSTC2, C_0 can contain up to three constraints: `food`, `area` and `pricerange`, with a sampled probability of 0.66, 0.62, and 0.58, respectively. Therefore, when generating the initial random goal, each constraint is subjected to the independent probability to determine whether it appears.

When training GUS, we did not use the goal generator, but the `goal-label` in DSTC2. DSTC2 provides turn-specific goal labels for each turn of the dialogue, which contain all constraints imposed by the user by the current turn.

When the system outputs the `canthelp` dialogue act, it means that there is no venue that matches the current user goal. The goal generator will regenerate a random goal G_T to replace the previous one.

3.2 Graph Generator

The graph generator constructs semantic graphs based on semantic inputs. For certain dialogue turn, the structure of the semantic graph is shown in Fig. 2. The whole graph is a tree structure with three layers. The motivation is to preserve all necessary value-independent information in a turn of dialogue.

The first layer contains only one node `turn_index`, whose feature (i.e., node value) is the index of the dialog turn. This node is the central node of the whole graph, which is used to store the aggregation of features of all other nodes in the current turn. Its node index is fixed to 0.

The second layer contains four nodes, which are `dialog_act` node, `requested_slot` node, `inconsistent_slot` node, and `user_goal` node respectively (Fig. 2 nodes 1-4). The features of these four nodes are fixed for any turn of dialogue, namely the strings `'dialog_act'`, `'requested_slots'`, `'inconsistency_slots'` and `'goal_labels'`. They are designed mainly to store the information of respective child nodes.

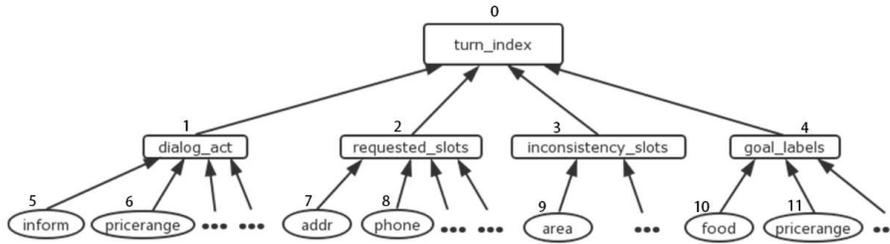


Fig. 2. Semantic graphs generated using certain turn of dialogue data

The child nodes of the dialog act node store information related to system acts, which are divided into two classes. The first class stores all dialog acts. Each act is a node by itself (Fig. 2 node 5). The second class stores slot information related to system acts (Fig. 2 node 6). When system act is `request` or `select`, or is `inform` or `expl-conf` with a correct pair of informable slot-value, a node is created for each corresponding slot. The motivation is that the difference in user responses required for different slots is often large for the same system action. For example, the structure of the user utterance that responds to `request (food)` and `request (area)` can be quite different.

The child nodes of the requested slot node store all request information that has not been fulfilled (Fig. 2 nodes 7,8). In DSTC2, user requests are stored in `requested-slots`, from which we can directly construct nodes. When the system act is `inform` or `offer`, the slot corresponding to the system act will be marked as fulfilled, and then be removed.

The child nodes of the inconsistent slot node store the inconsistency between the user goal and the system response (Fig. 2 node 9). When system act is `inform` or `expl-conf` or `impl-conf`, the system will propose one or more informable slots accordingly. Some of these slots may not appear in the user goal, which are added to the branch. The storage of inconsistent information is necessary, which helps GUS to correct the system.

The child nodes of the user goal node store all constraints contained in the current user goal (Fig. 2 nodes 10,11). From the turn-specific goal label in DSTC2, all constraint information can be directly obtained.

3.3 GCN Feature Extractor

To encode node features in the semantic graph, GUS adopts the GCN architecture. The purpose of GCN is to calculate the representation of each node in the graph structure [18]. Formally, given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where \mathcal{V} is the set of all vertices, \mathcal{E} is the set of all edges. For each node $v_i \in \mathcal{V}$ in the graph, use $h_i^{(l)}$ as the representation of v_i at the l th layer of GCN, then $h_i^{(l+1)}$ is [17]:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} h_j^{(l)} W^{(l)} + b^{(l)}\right) \quad (1)$$

Where \mathcal{N}_i is the set of adjacent nodes of v_i . c_{ij} is the normalized constant corresponding to the edge (v_i, v_j) . $W^{(l)}$ is the weight matrix of layer l . $b^{(l)}$ is the bias of layer l . σ is a nonlinear differentiable activation function (ReLU). As pointed out by Marcheggiani [19], a self-loop should be added for each node in the graph to ensure that $h_i^{(l)}$ will affect the generation of $h_i^{(l+1)}$. $h_i^{(0)}$ is the initial node feature of v_i in \mathcal{G} .

Considering that the semantic graph is a three-layer tree structure, and the information of all nodes in the third layer needs to be aggregated into the central node in the first layer (i.e., node 0), GCN is designed as two layers. The initial representation of each node is the word embedding of node features. After two-layer convolution, node 0 stores the information of the entire graph. So we use the last representation of node 0 as the feature vector of the graph. That is, for semantic graph g_T , its corresponding feature vector \mathbf{v}_T is:

$$\mathbf{v}_T = h_{0_T}^{(2)} \quad (2)$$

Where 0_T is node 0 of graph g_T . Each turn of dialogue generates its own feature vector, which is combined in a turn order as the feature history $\mathbf{v}_{1:T} = \mathbf{v}_1 \dots \mathbf{v}_T$. It will then be inputted into the sequence-to-sequence model.

3.4 Sequence-To-Sequence Model

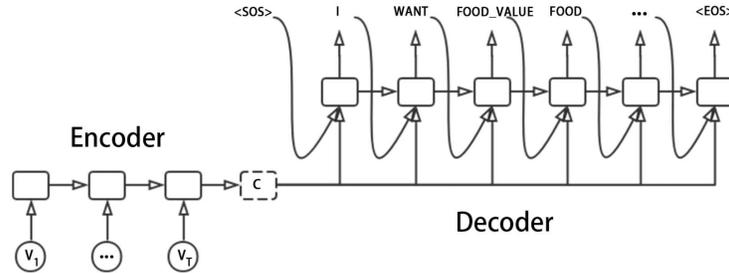


Fig. 3. The sequence-to-sequence model in GUS

The sequence-to-sequence model contains an RNN encoder and an RNN decoder, as shown in Fig. 3. The definition of RNN is as follows:

$$(\mathbf{h}_t, \mathbf{s}_t) = RNN(\mathbf{x}_t, \mathbf{s}_{t-1}) \quad (3)$$

At time step t , RNN takes the input \mathbf{x}_t and the last hidden state \mathbf{s}_{t-1} , outputs \mathbf{h}_t and the new hidden state \mathbf{s}_t . RNN contains many variants. In GUS, both the encoder and decoder use the single-layer and single-direction LSTM [20].

The encoder takes one feature vector as input at each time step t , i.e. $\mathbf{x}_t^E = \mathbf{v}_t$. If the current turn is T , then the final output of the encoder is \mathbf{h}_T^E , which is also called the context vector (Fig. 3 \mathbf{c} in the dotted box).

The sequence-to-sequence model must define a probability distribution over different sequences on \mathbf{c} so that when GUS is sampled on this distribution, different sets of utterances can be generated from the same context. The conditional probability distribution of a length L sentence is defined as:

$$P(\mathbf{u}|\mathbf{c}) = P(\omega_0|\mathbf{c}) \prod_{t=1}^L P(\omega_t|\omega_{t-1}\dots\omega_0, \mathbf{c}) \quad (4)$$

The decoder is used to model $P(\omega_t|\omega_{t-1}\dots\omega_0, \mathbf{c})$. Its input at a time is the concatenation of the word embedding \mathbf{w}_{t-1} of the previous step output word ω_{t-1} and the context vector \mathbf{c} , that is, $\mathbf{x}_t^D = [\mathbf{w}_{t-1}\mathbf{c}]$. For $P(\omega_0|\mathbf{c})$, the sentence start tag <SOS> is used as ω_{-1} . When the decoder outputs the sentence end tag <EOS>, the decoding process is finished. The output \mathbf{h}_t^D forms $P(\omega_t|\omega_{t-1}\dots\omega_0, \mathbf{c})$ through a projection layer defined as follows:

$$P(\omega_t|\omega_{t-1}\dots\omega_0, \mathbf{c}) = \text{Softmax}(W_\omega \mathbf{h}_t^D + \mathbf{b}_\omega) \quad (5)$$

The word ω_t is obtained by taking the word with the highest probability from the distribution. During the training of GUS, ω_t is not used as the input of the next time step, but directly using the ground-truth words in the dataset. This technique is called teacher-forcing. In addition, the user utterances in the dataset are delexicalized. It can be seen in Fig. 3 that the value in the output utterance of the decoder has been replaced with the corresponding slot.

The training goal of the sequence-to-sequence model is to maximize the output sequence probability conditioned on the input sequence, so the log probability corresponding to formula (4) can be used as the loss function:

$$\mathcal{L} = - \sum_{n=1}^N \log P(\omega_0|\mathbf{c}) \sum_{t=1}^{L_n} P(\omega_t|\omega_{t-1}\dots\omega_0, \mathbf{c}) \quad (6)$$

4 Experimental Setup and Results

4.1 Experimental Setup

Dataset GUS is trained and tested on the DSTC2 dataset, which is collected by Amazon Mechanical Turk on the restaurants recommendation domain. The dialogue state in DSTC2 is very variable, and users often change their goals during the dialogue. DSTC2 is manually recorded, so it contains many spelling and recording errors. We corrected some common spelling errors to ensure correct delexicalization. The training set contains 2,118 dialogues, and the test set contains 1,117 dialogues. After delexicalization, the maximum sequence length is 22. The maximum dialogue length is 30 turns.

Evaluation Method The evaluation of the user simulator is an ongoing research area, and various technical methods can be found in related work. We

use BLEU [22] (Bilingual Evaluation Understudy) and real dialogue output to measure the performance.

BLEU is a method for automatic evaluation of machine translation. BLEU has been widely employed in evaluating sequence generation including machine translation, text summarization and dialogue systems. BLEU calculates the n-gram precision which is the fraction of n-grams in the candidate text present in any of the reference texts.

In addition to BLEU, we also verify the validity of the model by establishing a real dialogue history and observing the model output utterance. This is a relatively straightforward way to evaluate the model.

Baseline and Setup All models use the full teacher-forcing mode and maximum decoding steps is 15. The experimental setup is in Table 1.

- GCEUS: This model directly uses GCN as the encoder and discards the original LSTM structure. The encoder combines the semantic graphs of all turns into a large graph, and uses the representation of the central node after two-layer convolution as the final output. The decoder is the same as GUS.
- S2SUS: The semantic input data is sequenced and directly inputted into the sequence-to-sequence model, which omits the step of extracting the feature vector.
- NUS: This model is proposed by Kreyssig [1]. It outperforms the classic ABUS and is the reference architecture of GUS. It extracts 35-dimensional feature vectors from the dialog data based on hand-crafted rules.

Table 1. Experimental setup of all models

Model name	Encoder input dimension	Decoder output dimension	Hidden size	Batch size	Learning rate
GCEUS	256	256	512	64	0.0004
S2SUS	256	256	512	32	0.0003
NUS	35	128	256	16	0.001
GUS	256	256	512	64	0.0007

4.2 Results

As Table 2 shows, the BLUE of GUS is 0.211, which is the highest among all models. It reflects that GUS can generate user utterances that best fit the context and contain the necessary information. Table 3 shows the user utterance that all models output on the same dialogue history. The history was not taken from DSTC2, but was artificially constructed. Since all models can only accept input in semantic form rather than natural language, the dialogue history was converted into json data of a specific format, which contained the semantics of

Table 2. The highest BLEU of all models on test set after 20 epochs training

Model name	BLEU
GUS	.211
NUS	.169
S2SUS	.136
GCEUS	.099

Table 3. Outputs of all models on the same real dialogue history

Dialogue history	System: How may I help you? User: I'm looking for food_value food in the area_value part of town. System: There are 12 restaurants in the west of town. What type of food do you want? User: food_value food. System: There are 4 restaurants serving Indian in the west of town. What price range would you like?
Model name	Outputs
GUS	pricerange_value
NUS	what is the address
S2SUS	food_value
GCEUS	address

each turn of dialogue. The data was then inputted into each model. In the final turn of the dialogue, the system asked the user what price range he liked. GUS correctly answered the price range. NUS requested the address of the restaurant, which was obviously wrong. S2SUS incorrectly answered what food to eat. GCEUS was the same as NUS which also asked for the restaurant address. Only GUS answered with the correct user utterance, which verifies its validity again. In both evaluation tasks, GUS works best for all models. In order to verify that it is able to finish a complete and successful dialogue, we asked a human as the system to have a real dialogue with GUS. The dialogue is shown in Table 4. In the 4th turn, the system said there was no venue that met the requirements, and GUS successfully changed the user goal in the 5th turn and began to re-inform the system of its constraints on the new venue. When the system proposed a venue in the 7th turn, GUS asked for the address, phone number and zip code in order. When the system did not correctly inform the phone number in the 9th turn, GUS re-inquired in the 10th turn. Finally, in the 12th turn, GUS determined that the task had been completed, and output the thank-you statement. The dialogue ended. It can be seen that GUS can output context-compliant user utterances in practical applications.

4.3 Analysis

First, the BLEU of all models is relatively low. The reason is presumed as follows: All models are trained on DSTC2, which is a very noisy dataset containing many spelling errors. Although the commonly occurred errors are corrected, there are

Table 4. Dialogue between GUS as the user and human as the system

Turn	GUS	Human
		how may I help you?
0	i'm looking for food_value food	what kind of food would you like?
1	food_value	you're looking for a restaurant serving chinese food, right?
2	yes	what part of town do you have in mind?
3	area_value	there are 3 restaurants in west of the town that serve chinese food. what price range do you want?
4	pricerange_value	I'm sorry but there is no expensive restaurant
5	food_value food	what area do you prefer?
6	area_value	and what price range would you like?
7	do nt care	golden house is a great restaurant that serves indian food
8	address	golden house is a nice place. The address is on Ohoho Street
9	phone number	golden house is a nice place. The address is on Ohoho Street
10	phone number	the phone number of golden house is 213124512
11	post code	the post code of golden house is A.7 B. C1
12	thank you	goodbye

also quite a few accidental errors. Besides, DSTC2 often misses or mislabels semantic data when annotating. These errors cannot be solved except for manual correction. In addition, each dialogue in DSTC2 has a corresponding customer feedback, many of which are marked as "disagree". But these noise dialogues are not removed. Furthermore, the amount of data is relatively small, which may also result in a decrease in the performance of the model on the test set. And as shown in Table 2 and Table 3, the user utterances output by the model are mostly very short. Although the necessary information is provided, this also leads to a lower BLEU.

Compared with NUS, GUS has a large improvement (from 0.169 to 0.211). This shows that the effect of feature extraction using GCN is better than that of hand-crafted rules. The semantic graph design of GUS refers to the design of the feature vector of NUS, both of which divides the semantics into four parts. In NUS, a simple binary vector is used to represent the semantics of each part, and the four parts are simply concatenated. In GUS, the four-part semantics are explicitly distinguished by the nodes in the middle layer. Through training, the corresponding weights are learned for the nodes on each branch, which is equivalent to a more elaborate combination of semantics and leads to a better effect.

S2SUS's BLEU is lower than GUS and NUS, which means that it is necessary to extract feature vectors from semantic data. There are some information that does not affect the result (such as user acts) in the input. Performing feature extraction operation on the semantic data can help the sequence-to-sequence model learn the input features better, thereby improving its effect.

The BLEU of GCEUS is the lowest among all models, which is even lower than S2SUS (0.136 to 0.099). It indicates that the GCN encoder is less effective

than the RNN encoder in tasks that require modeling dialogue history. Because the nature of dialogue is a sequence of data, the content of the last turn is significantly more important than the previous history, which coincides with the fact that the later input in the RNN encoder has a greater impact on the results. In the GCN encoder, the order of different turns is not explicitly distinguished, and relies on the network to learn by itself. So the final effect is very poor.

5 Conclusion

We propose the Graph Neural Net-based User Simulator (GUS) architecture, which takes the semantic response of the system as input and outputs user utterances in natural language form. Compared with Neural User Simulator (NUS), GUS uses GCN to extract feature vectors, which makes the extraction process more detailed and efficient, and helps the sequence-to-sequence model better learn the dialogue features. This work is the first time to introduce the GNN architecture to the field of user simulator. It not only explores the feasibility of GNN applications on semantic data, but also achieves good results. In the BLEU evaluation task, GUS has a great improvement than NUS, and is much higher than S2SUS and GCEUS. In the evaluation of establishing real dialogue history and observing output, GUS can correctly reply to system statements and complete a successful dialogue.

Acknowledgement

We thank the anonymous reviewers for their insightful comments on this paper. This work was supported by the NSFC (No. 61402403), Alibaba-Zhejiang University Joint Institute of Frontier Technologies, Chinese Knowledge Center for Engineering Sciences and Technology, and the Fundamental Research Funds for the Central Universities.

References

1. Florian Kreyszig, Inigo Casanueva, Pawel Budzianowski, Milica Gasic. Neural User Simulation for Corpus-based Policy Optimisation for Spoken Dialogue Systems. *Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue*, 2018
2. Milica Gasic, Steve Young. Gaussian Processes for Pomdp-based Dialogue Manager Optimization. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(1):28-40, 2014
3. Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, Maosong Sun. Graph Neural Networks:A Review of Methods and Applications. *arXiv preprint arXiv:1812.08434v3*, 2019
4. Jost Schatzmann, Kallirroi Georgila, Steve Young. Quantitative Evaluation of User Simulation Techniques for Spoken Dialogue Systems. *SIGdial6*, 45-54, 2005
5. Wieland Eckert, Esther Levin, Roberto Pieraccini. User Modeling for Spoken Dialogue System Evaluation. *Automatic Speech Recognition and Understanding, 1997 IEEE Workshop on*, pages 80-87, 1997

6. Konrad Scheffler, Steve Young. Probabilistic Simulation of Human-machine Dialogues. *Acoustics, Speech, and Signal Processing*, 2000
7. Olivier Pietquin, Thierry Dutoit. A Probabilistic Framework for Dialog Simulation and Optimal Strategy Learning. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(2):589-599, 2006
8. Kallirroi Georgila, James Henderson, Oliver Lemon. Learning User Simulations for Information State Update Dialogue Systems. *Ninth European Conference on Speech Communication and Technology*, 2005
9. Senthilkumar Chandramohan, Matthieu Geist, Fabrice Lefevre, Olivier Pietquin. User Simulation in Dialogue Systems Using Inverse Reinforcement Learning. *Proceedings of the Twelfth Annual Conference of the International Speech Communication Association*, 2011
10. Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, Steve Young. Agenda-based User Simulation for Bootstrapping a Pomdp Dialogue System. *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 149-152, 2007
11. Nikola Mrksic, Diarmuid O Seaghdha, Tsung-Hsien Wen, Blaise Thomson, Steve Young. Neural Belief Tracker: Data-driven Dialogue State Tracking. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1777-1788, 2017
12. Jason D Williams. Evaluating User Simulations with the Cramer-von Mises Divergence. *Speech Communication*, 50(10):829-846, 2008
13. Paul Crook, Alex Marin. Sequence to Sequence Modeling for User Simulation in Dialog Systems. *Proceedings of the 18th Annual Conference of the International Speech Communication Association*, 2017
14. Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, Asli Celikyilmaz. End-to-end Task-completion Neural Dialogue Systems. *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2017
15. Manex Serras, Maria Ines Torres, Arantza Pozo. Regularized Neural User Model for Goal Oriented Spoken Dialogue Systems. *International Workshop on Spoken Dialogue Systems*, 2017
16. Bing Liu, Ian Lane. Iterative Policy Learning in End-to-end Trainable Task-oriented Neural Dialog Models. *2017 IEEE Automatic Speech Recognition and Understanding Workshop, ASRU*, pages 482-489, 2017
17. Thomas N. Kipf, Max Welling. Semi-supervised Classification with Graph Convolutional Networks. *Proceedings of the International Conference on Learning Representations, ICLR*, 2016
18. D. Marcheggiani, L. Perez-Beltranchini. Deep Graph Convolutional Encoders for Structured Data to Text Generation. *The 11th International Conference on Natural Language Generation, INLG*, 2018
19. Diego Marcheggiani, Ivan Titov. Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1506-1515, 2017
20. Sepp Hochreiter, Jurgen Schmidhuber. Long Short-term Memory. *Neural computation*, 9(8):1735-1780, 1997
21. Diederik P Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization. *Proc. ICLR*, 2015
22. Papineni K., Roukos S., Ward T., Zhu W. Bleu: A Method for Automatic Evaluation of Machine Translation. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311-318, 2002