End-to-End Recurrent Entity Network for Entity-Value Independent Goal-Oriented Dialog Learning

Chien-Sheng Wu, Andrea Madotto*, Genta Indra Winata, Pascale Fung*

Human Language Technology Center
Department of Electronic and Computer Engineering
The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong
[cwuak, eeandreamad, giwinata]@ust.hk, pascale@ece.ust.hk

Abstract

This paper presents an end-to-end solution for the goal-oriented dialog system task in Dialog System Technology Challenges 6 (DSTC6). The challenge consists in learning a dialog policy from a given restaurant booking domain. End-to-end models are required to reason over dialog entities and to track the dialog states. Hence, we introduce a practical entity-value independent framework based on Recurrent Entity Networks. The framework is able to abstract linguistic entity by using a delexicalization mechanism, which improves the original model performance especially in test sets with out-of-vocabulary entities. Recurrent Entity Networks also plays an important role to represent the latent dialog state and the dialog policy. As shown in experiments, our framework can achieve a promising average Precision-1 of 96.56% in all the test sets.

Index Terms: Goal-oriented Dialog Systems, Memory Network, Recurrent Entity Networks, Delexicalization, Natural Language Processing

1. Introduction

Goal-oriented dialog requires skills more than language modelling, e.g., understanding user request, asking for clarification, properly issuing API calls, querying knowledge base (KB) and interpreting query results. Traditionally, these dialog systems have been built as a pipeline, with modules for language understanding, state tracking, action selection, and language generation [1, 2, 3, 4, 5]. Even though those systems are known to be stable via combining domain-specific knowledge and slot-filling technique, they have limited ability to generalize into new domains and the dependencies between modules are quite complex.

On the other hand, end-to-end approaches train model directly on text transcripts of dialogs, and learn a distributed vector representation of the dialog state automatically [6, 7, 8, 9]. In this way, models make no assumption on dialog state structure, holding the advantage of easily scaling up. Specifically, using recurrent neural networks (RNNs) is an attractive solution, where the latent memory of an RNN represents the dialog state. However, the hidden states (memories) may be inherently unstable over long time sequences. Several RNN structures have been proposed to overcome the problem [10, 11, 12], where the models are designed to represent long-term memories through global memory cells or gated functions. Highly related to our work, Recurrent Entity Networks [12] with dynamic long-term memory blocks have been demonstrated to have promising performance on reasoning and language understanding, which are also essential abilities for goal-oriented dialog learning.

In DSTC6 track 1 [13], end-to-end goal-oriented dialog learning tasks, we present a practical entity-value independent framework based on Recurrent Entity Network and a recorded delexicalization mechanism. The former can be seen as a bank of gated RNNs which all sharing the same parameter but distinct memory slots. The latter not only decreases the learning complexity but also makes our system scalable into new out-of-vocabulary (OOV) KB. We first introduce the dialog tasks in DSTC6 (Section 2) and outline the key methodologies we used (Section 3). Then the model settings, training details and the obtained results are showed (Section 4).

Table 1: *Dialog example of task 1 (Issue API call)*

	Utterance in Dialog
U	May I have a table for four in paris, with a romantic atmosphere and moderate price
S	I'm on it
U	<silence >
S	Any preference on a type of cuisine
U	My friend wants british, let's do that
S	Ok let me look into some options for you
U	<silence >
Candidates: *1. whenever you're ready* *2. api_call british paris four moderate romantic* ... *11. api_call british london four moderate romantic*	
Answer: *api_call british paris four moderate romantic*	

2. Task Description

The DSTC6 organizers expanded the original bAbI Dialog [14] dataset by using the same simulator. The challenge requires solving five tasks, each of which is corresponding to a particular dialog system ability such as issuing API calls, updating API calls, displaying options, providing extra information and conducting full dialogs. For training set, each task includes 10k dialogs, each of which has its corresponding system response candidates and the correct answer. An example of issuing API call is shown in Table 1.

The task requires to rank the candidates and will evaluate the model's performance by using Precisions 1,2,5 over four different test sets (with 1k samples each). Each test set includes the five different tasks mentioned. The test set 1 uses the same KB of the training set, instead test set 2 uses a new KB which

* These two authors contributed equally.

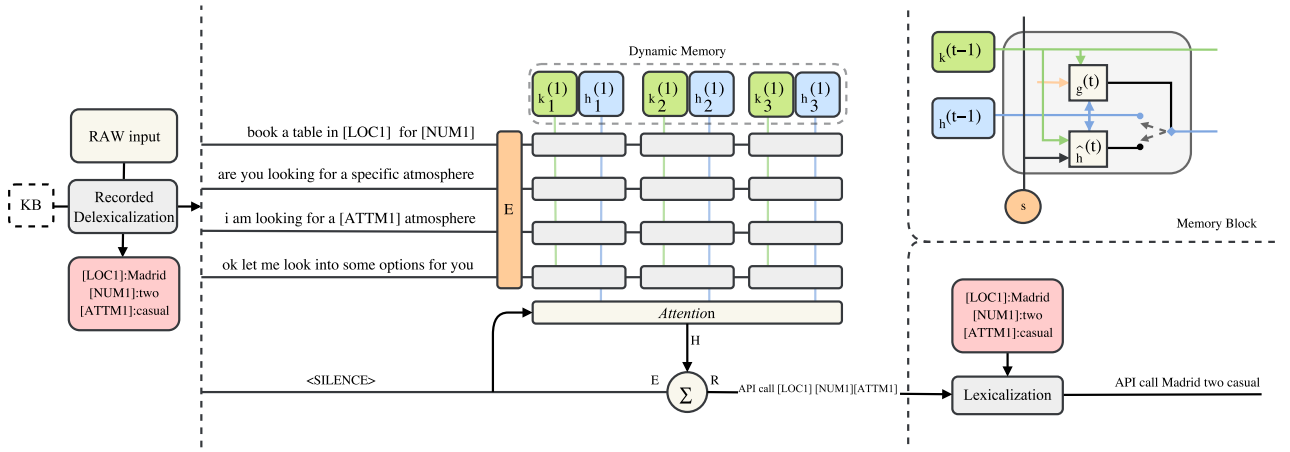

Figure 1: *Entity-Value Independent Recurrent Entity Network for goal-oriented dialog. The image on top right shows the detailed memory block.*

includes out-of-vocabulary (OOV) words. The test set 3 and 4 are required to fill an additional slot for the queries, but the former uses the same training KB and the latter uses the new KB. A more detailed description can be found in the article [13].

3. Model Description

Our framework is mainly made of two components: Recurrent Entity Network (REN) [12] and Recorded Delexicalization (RDL). The key improvement is given by the RDL, where the similar intuition can be found in other models [7, 8]. RDL decreases the learning complexity in particular while using REN, which is an excellent model for reasoning over abstract entities. Thus, REN outputs the next dialog utterance by choosing among action templates. The last step, lexicalization, simply replace delexicalized elements in action template with plain text based on a lookup table. The details are described below.

3.1. Recorded Delexicalization

We utilize existing KB information to extract entities from both user and system utterances. In restaurant domain, we extract nine entity types including *[NAME, LOCATION, PRICE, PHONE, CUISINE, ATMOSPHERE, RESTRICTIONS, NUMBER, ADDRESS]* using simple string matching. For example, we recognize Bombay, Rome, London, Paris and Madrid as location based on the KB information. However, we keep the real number of *[RATING]* in the utterances and let the REN to sort restaurant ranks. Then we replace each real entity value with its entity type and the order appearance in the dialog, and we also build a lookup table to record the mapping. For example, the first user utterance in Figure 1, "*book a table in Madrid for two*", will be transformed into "*book a table in [LOC1] for [NUM1]*". At the same time, *[LOC1]* and *[NUM1]* are stored in a lookup table as Madrid and two, respectively. At last, we build the action templates *act* by all the possible delexicalization system responses. Note that the action template can always return to a plain response by the inverse of RDL. For example, the output action template in Figure 1, "*api_call [LOC1][NUM1][ATTM1]*", will be lexicalized into "*api_call Madrid two casual*". One can consider RDL as an easy method of name entity recognition that only recognizes the entities defined in the KB and also take their order in dialog into account.

3.2. Recurrent Entity Network

The Recurrent Entity Network has three main components: *Input Encoder*, *Dynamic Memory*, and *Output Module*. Let's define the training data as a set of tuples $\{(x_i, y_i)\}_{i=1}^{n}$, with n equal to the sample size, where: x_i is a tuple (D, q) where $D = \{s_1, \ldots, s_{t-1}\}$ is the dialog history without the last sentence, and $q = s_t$ the last dialog sentence representing the "question". Instead, y_i is an action template (a possible system utterance) that represents the answer.

The Input Encoder transforms the set of words of a sentence s_t and the question q into a single vector representation by using a multiplicative mask. We define $E \in \mathbb{R}^{|V| \times d}$ the embedding matrix, i.e., $E(w) = e \in \mathbb{R}^d$, where d is the embedding size and V the vocabulary size. Hence, $\{e_i\}_{i \in s_t}$ are the word embedding of each word in the sentence s_t and $\{e_k\}_{k \in q}$ the embedding of the question's words. The multiplicative masks for the dialog sentences are defined as $f^{(s)} = \{f_i^{(s)}\}_{i \in s_t}$ and $f^{(q)} = \{f_k^{(q)}\}_{k \in q}$ for the question, where each $f_i \in \mathbb{R}^d$. The encoded vector of a sentence is defined as:

$$s_t = \sum_r e_r \odot f_r^{(s)} \qquad q = \sum_r e_r \odot f_r^{(q)}$$

The Dynamic Memory stores information of entities present in D. This module is very similar to a Gated Recurrent Unit (GRU) [16] with a hidden state divided into blocks. Moreover, each block ideally represents an entity (i.e. *LOC, PRICE* etc.), and it stores relevant facts about it. Each block i is made of a hidden state $h_i \in \mathbb{R}^d$ and a key $k_i \in \mathbb{R}^d$. The dynamic memory module is made of a set of blocks, which can be represent with a set of hidden states $\{h_1, \ldots, h_z\}$ and their correspondent set of keys $\{k_1, \ldots, k_z\}$. The equation used to update a generic block i are the following:

$$g_i^{(t)} = \sigma(s_t^T h_i^{(t-1)} + s_t^T k_i^{(t-1)}) \quad \text{(Gate Func.)}$$

$$\hat{h}_i^{(t)} = \text{ReLU}(U h_i^{(t-1)} + V k_i^{(t-1)} + W s_t) \quad \text{(Cand. Mem.)}$$

$$h_i^{(t)} = h_i^{(t-1)} + g_i^{(t)} \odot \hat{h}_i^{(t)} \quad \text{(New Mem.)}$$

$$h_i^{(t)} = h_i^{(t)} / \|h_i^{(t)}\| \quad \text{(Reset Mem.)}$$

where σ represents the sigmoid function, and ReLU is the Rectified Linear unit [17]. $g_i^{(t)}$ is the gating function which de-

Table 2: *Average Precisions 1 among all the tasks and test sets of the 5 model settings.*

Models	Precision@1	Precision@2	Precision@5
REN + RDL	.9576	.9734	.9911
REN + RDL + INFO	.9586	.9734	.9911
REN + RDL + INFO + POST	.9625	.9751	.9914
QDREN [15] + RDL + INFO + POST	.9618	**.9779**	**.9983**
REN + RDL + INFO + POST + DUMMY	**.9656**	.9765	.9862

termines how much of the ith memory should be updated, and $\hat{h}_i^{(t)}$ is the new candidate value of the memory to be combined with the existing $h_i^{(t-1)}$. The matrix $U \in \mathbb{R}^{d\times d}$, $V \in \mathbb{R}^{d\times d}$, $W \in \mathbb{R}^{d\times d}$ are shared among different blocks, and are trained together with the key vectors.

The Output Module creates a probability distribution over the memories' hidden states using the question q. Thus, the hidden states are summed up, using the probability as weight, to obtain a single vector representing all the input. Finally, the network output is obtained by combining the final state with the question. Let us define $R \in \mathbb{R}^{|act|\times d}$, $H \in \mathbb{R}^{d\times d}$, $\hat{y} \in \mathbb{R}^{|act|}$. Then, we have

$$
\begin{aligned}
p_i &= \text{Softmax}(q^T h_i) \\
u &= \sum_{j=1}^{z} p_j h_j \\
\hat{y} &= \text{ReLU}(q + Hu)
\end{aligned}
$$

The model is trained using a cross-entropy loss $H(\hat{y}, y)$, where y is the one hot encoding of the correct action template. Note that $\hat{y}$ can be viewed as the predicted scores of each action templates, that is, action template with the higher score is more likely to be the correct answer. All the parameters, including the embedding matrix, are learned using Backpropagation Through Time (BPTT) algorithm. A schematic representation of the model is shown in Figure 1.

4. Experiments

4.1. Training Details

For each task, we fix the number of the memory block to 5, and we used Adam [18] optimizer. The gradient was clipped to a maximum of 40 to avoid gradient explosion. 10% of training data is split into validation set. We have implemented an early stopping method, which stops the training ones the validation accuracy does not improve for 10 epochs. We try a small grid search over two hyperparameters such as batch size and embedding size. Then, the setting that achieved the highest accuracy in validation has been selected for the final evaluation. We use the $\hat{y}$ as score distribution to rank our candidates from 1 to 11. In addition, we augment the training data by learning not only on the original 10k data but also on the partial utterances in dialog history. Namely, every system response appearing in dialog history turns to be our training data. In this way, we efficiently increase our model performance due to data augmentation. Moreover, if the INFO feature in [14] is added, then speaker and temporal information are considered. For example, the second utterance in Table 1 will be "$S #2 I'm on it" under this setting.

4.2. Prediction Methods

During the prediction step, we face difficulties in the test set 3 and 4, which includes an additional slot that our model has never seen before. That is, those test sets include some candidates that never appear in our original action templates, and it is hard to evaluate their scores. To overcome this problem, we use a matching mechanism to find the most "similar" action template for the unfamiliar candidate. We simply compare the unfamiliar candidate word-by-word to each action template in act, then we find out the action template with the highest positional word matching, i.e., the one that looks more similar. For example, if "*api_call [LOC2][NUM2][ATTM4][UNK1]*" is a candidate not defined in act, then the matching action template is going to be "*api_call [LOC2][NUM2][ATTM4]*". Hence, we can rank this candidate based on the score of the matching action template.

However, one main drawback of the matching mechanism is that it assumes all the candidates are possible system responses since every unfamiliar candidate will be matched to one action template, even if those candidates are obviously user's utterances. Therefore, the DUMMY method adds all user utterances into act as dummy action templates. In this way, the unfamiliar candidate that is similar to user utterances may match to one of those dummy action templates, which is never considered as the right answer during training process. That is, it has a low score to become the correct response.

On the other hand, sometimes we may have the same score among candidates, which is possible because several unfamiliar candidates may match to the same action template. Here, we use the POST method if the only difference between those candidates is the additional slot. Based on our lookup table, we give a higher score to the candidate with latest slot value. For example, if both "*api_call [LOC2][NUM2][ATTM4][UNK1]*" and "*api_call [LOC2][NUM2][ATTM4][UNK2]*" are in the candidates, we prefer to choose the latter because *[UNK2]* appears after *[UNK1]*.

4.3. Results

The results of the 5 different settings[1] are summarized in Table 2. As we can see, the best model (REN + RDL + INFO + POST + DUMMY) achieves an average test Precision 1 of 96.56%. The other 4 settings are very close to the best one, with a margin less than 1%. Therefore, we can draw the following conclusions: (1) using temporal and user information (INFO) is not particularly useful for the REN since it already includes temporal dynamics by construction; (2) the post-processing step (POST) helps quite a lot (particularly for test set 3 and 4)[2] in order to match the correct API call with the additional slot; (3) QDREN model [15], which has the same architecture as REN but uses the last sentence in the memorization process, shows a

[1]The official evaluation is limited to 5 different settings.

[2]The precision of each test set is not showed due to the space limit

Table 3: *Precisions 1,2,5 of the our REN model with RDL + INFO + POST + DUMMY.*

		Task					
		T1	**T2**	**T3**	**T4**	**T5**	**Avg.**
Test1	*@1*	.983	.949	.998	.999	.986	.983
	@2	.992	.981	1.0	1.0	1.0	.995
	@5	.999	.996	1.0	1.0	1.0	.999
Test2	*@1*	.974	.958	1.0	.999	.991	.984
	@2	.992	.983	1.0	1.0	.999	.995
	@5	1.0	.996	1.0	1.0	1.0	.999
Test3	*@1*	.812	.967	.999	1.0	.950	.946
	@2	.846	.984	1.0	1.0	.959	.958
	@5	.899	.995	1.0	1.0	.967	.972
Test4	*@1*	.837	.96	.999	.998	.953	.949
	@2	.859	.976	1.0	1.0	.959	.959
	@5	.909	.996	1.0	1.0	.966	.974
Avg.	*@1*	.901	.956	.999	.999	.970	
	@2	.922	.981	1.0	1.0	.979	
	@5	.952	.996	1.0	1.0	.983	

more robust result and achieves the best Precision 2 and 5; (4) adding dummy user utterances in *act* (DUMMY) helps to select the correct system answer among the 11 candidates to rank. Our code is available here[3].

Furthermore, we report the detailed precisions of our best model in Table 3. Here we can notice that the average Precision 1 of the test set 3 and 4 is about 3.6% lower compared to test set 1 and 2. The precision drops especially in task 1, which is required to fill an additional slot. For instance, if a dialog system needs one more slot information about user's dietary restrictions, our model has limited ability to output the response "*do you have any dietary restrictions*". Instead, the response "*ok let me look into some options for you*" will score the highest because it is the correct response in our training data.

On the other hand, our model can handle well task 3 and 4 (i.e. recommending a restaurant and providing additional information) in all the test sets, where the average Precision 1 is 99.9%. This is largely due to the combination of REN and the RDL. Indeed, those tasks require focussing on particular entities in the input. By abstracting such entities, REN can focus on the correct abstract type.

4.4. Analysis

To better understand the REN behavior, we visualize the gating activation function in Figure 2. The output of this function decides how much and what we store in each memory cell. Moreover, we take the model trained on task 4 (i.e. providing additional information) for the visualization. We plot the activation matrix of the gate function and observe how REN learns to store relevant information.

As we can see in the figure, the model tries to open the memory gate once a useful information appears as input, and close the gate for other useless sentences. Different memory blocks may focus on different information. For examples, block 5 stores more information from the discourse rather than explicit KB knowledge; block 2 open its gate fully when the address and rating information is provided. In this case, the last user utterance (question) is "can you provide address", we can get the correct prediction because the latent address feature is

[3]https://github.com/jasonwu0731/RecurrentEntityNetwork

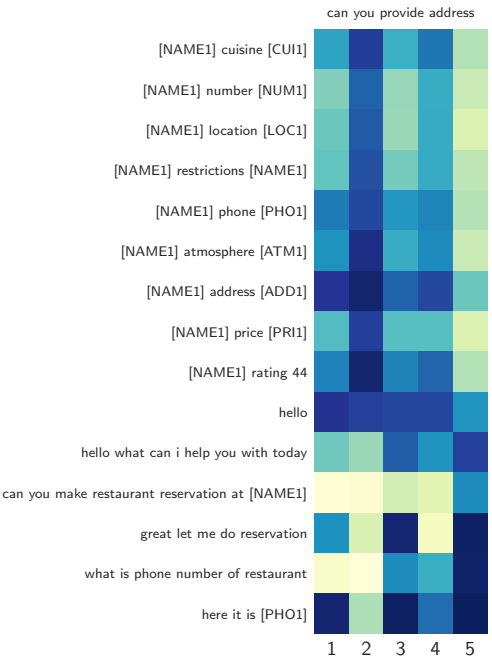

Figure 2: *Heatmap representing the gating function result for each memory block. The x-axis represents the memory block number, and in the y-axes, there are the utterances in the dialog divided into time steps, and at the top, there is the last user's utterance. Darker color means a gate more open (values close to 1) and lighter colour means the gate less open.*

represented in those memory blocks that open during the utterance "*[NAME1] address [ADD1]*".

5. Conclusion

In this paper, we showed the effectiveness of Recurrent Entity Network in modelling goal-oriented dialogs by using the DSTC6 dataset. Especially, this architecture performs well in combination with the recorded delexicalization mechanism and several simple prediction methods. REN shows an excellent ability to reason over entities, and the RDL highly reduces learning complexity and alleviates OOV problems. The average Precision-1 achieved by our model in all the test sets is promising (96.56%), but still, other competitors achieved 100% accuracy. However, we would like to highlight that our model is as end-to-end as possible, holding the promise of scaling up easily. Our future work is to design a more domain-general framework that can not only handle additional slots but also the out-of-domain setting.

6. References

[1] O. Lemon, K. Georgila, J. Henderson, and M. Stuttle, "An isu dialogue system exhibiting reinforcement learning of dialogue policies: generic slot-filling in the talk in-car system," in *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics: Posters & Demonstrations*. Association for Computational Linguistics, 2006, pp. 119–122.

[2] Z. Wang and O. Lemon, "A simple and generic belief tracking mechanism for the dialog state tracking challenge: On the believability of observed information." in *SIGDIAL Conference*, 2013, pp. 423–432.

[3] J. D. Williams and S. Young, "Partially observable markov deci-

sion processes for spoken dialog systems," *Computer Speech & Language*, vol. 21, no. 2, pp. 393–422, 2007.

[4] C. Hori, K. Ohtake, T. Misu, H. Kashioka, and S. Nakamura, "Statistical dialog management applied to wfst-based dialog systems," in *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*. IEEE, 2009, pp. 4793–4796.

[5] S. Young, M. Gašić, B. Thomson, and J. D. Williams, "Pomdp-based statistical spoken dialog systems: A review," *Proceedings of the IEEE*, vol. 101, no. 5, pp. 1160–1179, 2013.

[6] I. V. Serban, A. Sordoni, Y. Bengio, A. C. Courville, and J. Pineau, "Building end-to-end dialogue systems using generative hierarchical neural network models." in *AAAI*, 2016, pp. 3776–3784.

[7] J. D. Williams, K. Asadi, and G. Zweig, "Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning," in *ACL*, 2017.

[8] T. Zhao, A. Lu, K. Lee, and M. Eskenazi, "Generative encoder-decoder models for task-oriented spoken dialog systems with chatting capability," in *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*. Association for Computational Linguistics, August 2017, pp. 27–36. [Online]. Available: http://aclweb.org/anthology/W17-5505

[9] I. V. Serban, A. Sordoni, R. Lowe, L. Charlin, J. Pineau, A. C. Courville, and Y. Bengio, "A hierarchical latent variable encoder-decoder model for generating dialogues." in *AAAI*, 2017, pp. 3295–3301.

[10] S. Sukhbaatar, J. Weston, R. Fergus *et al.*, "End-to-end memory networks," in *Advances in neural information processing systems*, 2015, pp. 2440–2448.

[11] M. Seo, S. Min, A. Farhadi, and H. Hajishirzi, "Query-reduction networks for question answering," *ICLR*, 2017.

[12] M. Henaff, J. Weston, A. Szlam, A. Bordes, and Y. LeCun, "Tracking the world state with recurrent entity networks," *ICLR*, vol. abs/1612.03969, 2016.

[13] Y.-L. Boureau, A. Bordes, and J. Perez, "Dialog state tracking challenge 6 end-to-end goal-oriented dialog track."

[14] A. Bordes and J. Weston, "Learning end-to-end goal-oriented dialog," *ICLR*, vol. abs/1605.07683, 2017.

[15] A. Madotto and G. Attardi, "Question dependent recurrent entity network for question answering," *NL4AI*, 2017.

[16] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014, p. 103111.

[17] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[18] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *RCLR*, 2015.