End-to-End Goal-Oriented Dialog Learning Based On Memory Network

Byoungjae Kim[1], *KyungTae Chung*[2], *Jeongpil Lee*[1], *Jungyun Seo*[1], *Myoung-Wan Koo*[1]

[1]Sogang University, Korea
[2]Alluser.net Corp., Korea

wiz3021@sogang.ac.kr, chochobo@gmail.com, koreanfeel@gmail.com, seojy@sogang.ac.kr, mwkoo@sogang.ac.kr

Abstract

A model to fulfill the requirements of Dialog System Technology Challenges 6 (DSTC6) Track 1 was developed, where the task was to build end-to-end dialog systems for goal-oriented applications. This involved learning of a dialog policy from transactional dialogs of a given domain. Automatic system responses were generated using given task-oriented dialog data [http://workshop.colips.org/dstc6/index.html]. As this task has a similar structure to the question answering task [Weston, Bordes et al 2015], we employed the MemN2N architecture [Sukhbaatar et al 2015] in our model, because it outperforms other networks based on recurrent neural networks or long short-term memory. Problems arose when applying this model to the DSTC6 task. First, the original model yields an answer in word form, but a sentence-form answer was required for the DSTC6 tasks. Thus, we added an additional output memory representation ("D_layer") to the original model. Second, an out-of-vocabulary problem was encountered, which we resolved by changing the types of words that did not exist in the knowledge base. Finally, we developed a word-level prediction architecture for memory networks.

Index Terms: End-to-end goal-oriented dialog, human-computer interaction, memory network

1. Introduction

An end-to-end goal-oriented dialog learning task was set as part of the sixth Dialog System Technology Challenge (DSTC6) [1], which required end-to-end goal-oriented dialog learning that finds the correct answer in sentence form for a question in dialog. In this task, 10 answer candidates provided in a dialog are ranked based on their probability of being the correct answer. There were four test sets (hereafter abbreviated as "Tst") in the DSTC6 task: Tst 1 used the same knowledge base (KB) as the training set, and the same set of slots in the queries; Tst 2 used a second KB (with disjoint sets of phones, addresses, restaurants, locations, price, etc.), but the same set of slots in the queries; Tst 3 used the same KB as for the training set, but with one additional slot for the queries; and Tst 4 used the second KB and an additional required slot. Note that the system can access the first and the second KB.

In this study, we propose a slightly modified model of the memory network solving the question answering (QA) task [2] in order to perform the DSTC6 tasks. Note that the original model has a system that chooses an answer word among the entire answer words. This causes the first problem encountered in this approach, as the answer required for the DSTC6 task must be in sentence form. Thus, the proposed model chooses an answer sentence among all possible candidates using the method previously introduced by Bordes et al [3], as discussed in Sections 2 and 3. Further, we add an additional layer called the D_layer. We believe that one output layer is insufficient to represent a sentence, because the output layer is only used to represent one answer word. Therefore, we add one more output layer, which is expected to yield a higher-precision answer sentence and KB information. However, the experimental results indicate that our hypothesis is incorrect, as the performance degrades when the D_layer is employed. That is, the answer accuracy is slightly decreased.

2. Related Works

With regard to the architecture appropriate to end-to-end goal-oriented dialog learning, Ref. [3] describes four methods, employing term frequency–inverse document frequency (TF-IDF) matching, nearest neighbors, supervised embedding models (where embeddings are trained with a margin ranking loss), and memory networks.

Of the four methods, TF-IDF matching exhibits the lowest performance by a large degree. The nearest neighbor and supervised embedding methods exhibit similar performance for tasks 2 and 3 (T3; to provide application program interface (API) information). However, the supervised embedding models exhibit considerably better performance for T1 (issuing API calls), T4 (updating the calls with new information), and T5. Although supervised embedding models perform well for T1, this approach fails to yield the correct result for API calls to display options, T3, and T4. However, memory networks outperform across all tasks. They successfully perform T1 and T2 and provide improved results for the other tasks (but do not perfectly solve them). For T3–5, it is preferable to use 2 or 3 hops. For T3, use of 1 hop gives 64.8% accuracy while 2 hops give 74.7% accuracy. Further, memory networks with match type features give better performance. T4 becomes solvable because matches can be made to the results of the API call, and the out-of-vocabulary (OOV) results are improved. On the other hand, this approach fails to perform T3 and T5, and the performance is slightly decreased in T2. Note that the data used in Ref. [3] for T1–5 are all generated using the same simulator and share the same KB.

As memory networks exhibited the best performance in Ref. [3], they were selected for the model proposed in this study. Sukhbaatar et al [4] presents a well-modified version of the memory network proposed by Weston et al [5]. Single- and triple-layer versions of the model are shown in Figs. 1 and 2. Here, x_i are the sentences of the dialog, where the question q is the last sentence of the dialog. The answer $\hat{a}$ is an answer word. A sentence is a sum of word vectors. In the triple-layer

version, weighted matrices A^1, A^2, and A^3 are shared. Matrices C^1, C^2, and C^3 are shared in the same way. More detailed explanation is provided in Section 3.

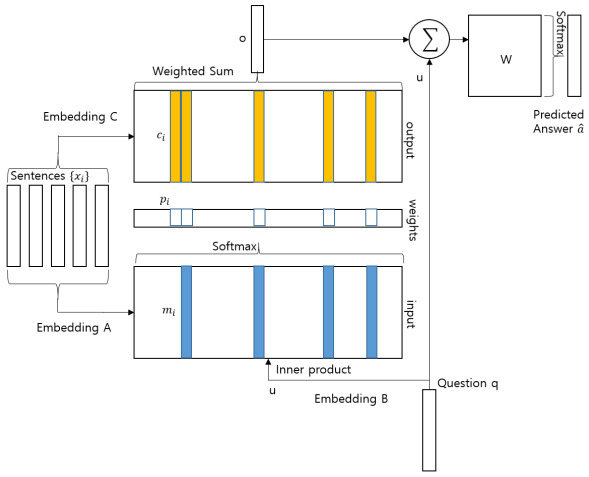

Figure 1: *Single-layer version of memory network model.*

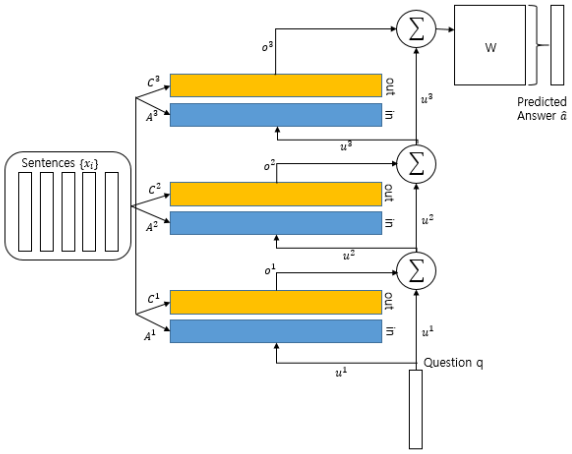

Figure 2: *Triple-layer version of memory network model.*

This memory network provides good results for QA problems [2] and also works well as a language model.

3. Proposed Model

We propose a model to solve the DSTC6 Track 1 tasks, i.e., end-to-end goal-oriented dialog learning that finds the correct answer in sentence form for a question in dialog. There are five tasks: T1: issuing API calls; T2: updating API calls; T3: displaying options; T4: providing additional information; and T5: conducting full dialogs. This model works appropriately for all tasks.

As noted above, we propose a model that chooses an answer sentence among all possible candidates, which is the same approach as used in Ref. [3]. In the training set, there are many similar answer and candidate utterances. Therefore, we can apply this method to our model. We attempt to predict an answer sentence using word-level features. In Ref. [3], the experimental results indicate that the match type is important to performance. Therefore, we employ a similar method. We change the slot words to a regular form, e.g., <R_rating.1>, <R_number.1>, <R_phone.1>, <R_address.1>, <R_name.1>, <R_location.1>, <R_price.1>, <R_cuisine.1>, <R_atmosphere.1>, and <R_restrictions.1>. Here, "R" indicates restaurant and ".1" indicates the first slot word that exists in the context, question, and answer. The second slot word is represented as <R_name.2>. The same word has the same form in a dialog. We refer to these altered forms as meta data type and save the slot-meta matching information in dictionary form to recover the original text word. Using this method, we can obtain considerably improved performance, comparing to not using the meta data type.

3.1. Sentence-level answer model with two output layers

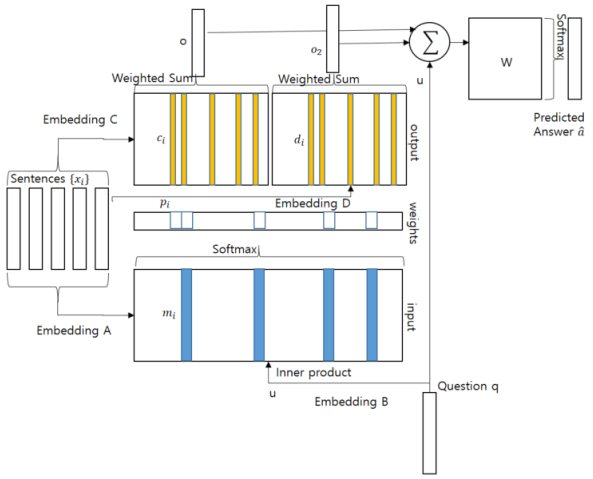

Figure 3: *Single-layer version of proposed model.*

The single-layer model architecture is shown in Fig. 3. We use a two-output-layer representation model (i.e., with the additional D_layer), because we believe this model can represent sentence information well. However, this model exhibits lower performance than a one-output-layer model, which we failed to rectify, as explained in Section 4. We employ a triple-layer version for all tasks; the model architecture is shown in Figure 4.

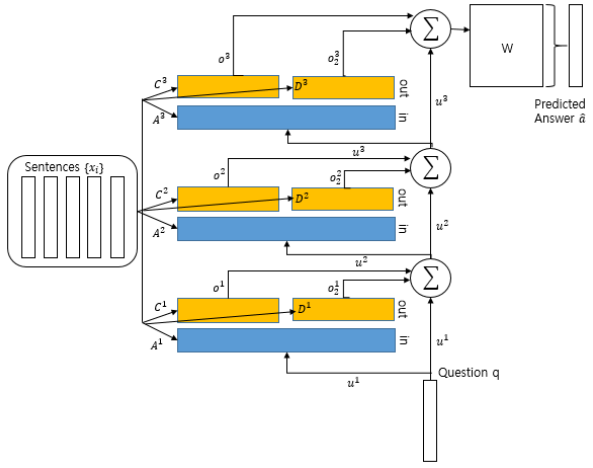

Figure 4: *Triple-layer version of proposed model.*

In this model, the embedding matrices (*A, B, C,* and *D*) are randomly initialized matrices. These are word embedding vectors initialized to a random normal distribution. A sentence

vector consists of a sum of word vectors. Embeddings A, C, and D are learned during training time. The input m_i and output c_i embedding equations are as follows:

$$m_i = Ax_i + T_A(i), \quad (1)$$

$$c_i = Cx_i + T_C(i), \quad (2)$$

$$d_i = Dx_i + T_D(i). \quad (3)$$

Here, $T_A(i)$ is the i th row of matrix T_A, which encodes temporal information. T_C and T_D are temporal matrices. These temporal matrix parameters are also randomly initialized and learned during training time.

The weighted vector p_i and embedded query vector u shown in Fig. 3 are defined as follows:

$$u = Bq\ , \quad (4)$$

$$p_i = Softmax(u^T m_i). \quad (5)$$

The output memories o and o_2 are weighted sums of c_i or d_i, weighted by the probability vector p_i, such that

$$o = \sum_i p_i c_i, \quad (6)$$

$$o_2 = \sum_i p_i d_i. \quad (7)$$

The final output, i.e., the predicted answer $\hat{a}$, is the sum of the output vectors o and o_2, and the input embedding vector u, which is passed through a weighted matrix W and softmax:

$$\hat{a} = Softmax\big(W(o + o_2 + u)\big). \quad (8)$$

In the triple-layer model version, embedding parameters are shared like recurrent neural networks (RNN), i.e., $A^1 = A^2 = A^3$, $C^1 = C^2 = C^3$, and $D^1 = D^2 = D^3$. The input to the layers is the sum of the outputs o^k and o_2^k, and the input u^k from layer k:

$$u^{k+1} = u^k + o^k + o_2^k. \quad (9)$$

The $\hat{a}$ of our model is a sentence, which is predicted from among all candidates in the training set. This method is selected because there is a very small out of answer (OOA) component. OOA refers to the number of answer candidates that exist in the test set, but do not exist simultaneously in the training set. We divide the training set into training, validation, and test sets. The new training set constitutes 85% of the original training set, with the test set corresponding to the remaining 15%. The validation set constitutes 20% of the new training set.

3.2. Meta data type features

The OOA and OOV ratios between the candidates in the training and test sets are listed for each task in Table 1.

Table 1: *OOA and OOV ratio between candidates in training and test sets.*

Task	– Meta data type		+ Meta data type	
	OOV	OOA	OOV	OOA
1	0	0	0	0.07
2	0	0	0	0.012
3	0.0037	0.007	0	0
4	0.026	0.026	0	0
5	0.022	0.02	0.014	0.3

To solve the OOV problem, we use meta data type features. Examples of the meta data types employed in our method are presented in Table 2.

Table 2: *Meta data type example.*

I prefer <R_cuisine.1> to <R_cuisine.2> cuisine.
I want a <R_price.2> price range, but my friend wants a <R_price.1> range. So, let's do that.
"R_cuisine": {"1": "Indian", "2": "British", "3": "Spanish"}
"R_price": {"1": "expensive", "2": "cheap"}

We adjusted the sentences to have forms such as those listed in Table 2. A dictionary of information matching the data types was established, as indicated at the bottom of Table 2. Through use of this method, the OOV ratio decreased by 1–2% for T4 and T5. Further, the OOV ratio of context utterances decreased by 40–50%. However, the OOA ratio increased by approximately 28% for T5, and increased slightly for T1 and T2. This was because the meta data content and numbers changed for each story and, therefore, the same word could be assigned to different forms. For example, "British" can correspond to <R_cuisine.1> or <R_cuisine.2>. Despite this disadvantage with regard to the OOA ratio, the model using meta data type features exhibited a performance improvement of approximately 18% for T5, compared to the model without this technique. We selected the variable words based on the provided extended KB 1 and 2 files, and added some rules to detect words similar to KB words.

3.3. Word-level answer model

In the word-level answer model, each A_1–$A_n(\overline{A_1}$–$\overline{A_n})$ score is generated through a softmax layer. Using this model, we can obtain the word included in the predicted answer. We can find a target answer through knowledge of the word inclusion, because the answer candidates are unique. Although we expected that the model with meta data type features would yield a higher-accuracy answer prediction, because the OOV ratio is much less than OOA ratio for T5, the result had lower accuracy. The top-1 answer precision was approximately 0.68 for T5; therefore, we rejected this architecture and decided to add a word feature layer, as shown in Fig. 6. We considered the use of pretrained embedding vectors instead of randomly initialized vectors. The word vectors were pretrained with GloVe [7], using the DSTC6 training set. However, the precision decreased, and the initial loss increased considerably.

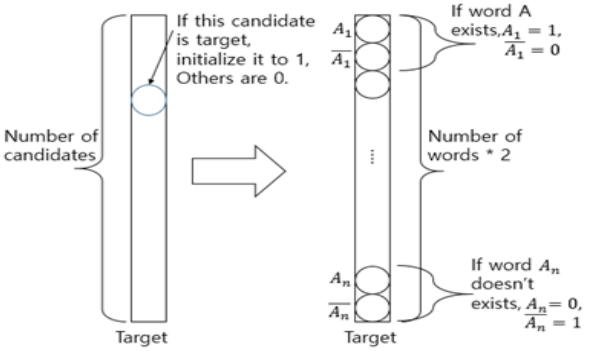

Figure 5: *Word-level target structure.*

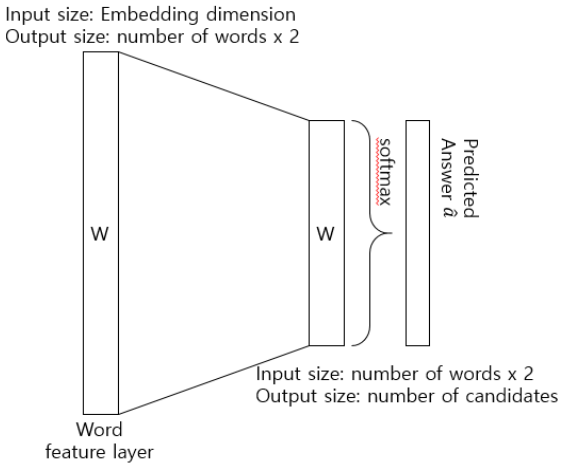

Figure 6: Word feature layer of proposed model.

4. Experiment

For evaluation, we divided the training data into three parts. The training set was 64% of the provided DSTC6 training set, the validation set was 21%, and the test set was 15%. The hyperparameters were as follows: embedding dimension: 20; layer number: 3; batch size: 32; initial learning rate: 0.005. The learning rate annealed by half every 25 epochs until 100 epochs were reached. Momentum or weight decay were not used. We used the linear start learning technique introduced by Sukhbaatar et al [4], which is initiated when the softmax of weight p_i is removed. During training, when the validation loss stops decreasing, the softmax is inserted. We performed experiments for each task using five-fold cross-validation. The responses consisted of 10 candidates, which we ranked from 1 to 10, with rank 1 indicating the highest probability of being the correct answer.

Table 3: *Average precision of top-1 answer.*

Task	One-output-layer model		Two-output-layer model	
	– meta	+ meta	– meta	+ meta
1	0.921	0.981	0.922	0.981
2	0.932	0.996	0.906	0.995
3	0.852	0.998	0.849	0.997
4	0.763	1.0	0.761	1.0
5	0.746	0.924	0.746	0.922

The results listed in Table 3 were poor, with precision decreasing for T2–5. Further, the precision fluctuated by approximately 2%. The best test scores obtained using our model are presented in Table 4, which were achieved using the two-output-layer model with meta data type features. In the DSTC6 challenge, our team (team 6) ranked in the second group (rank 5). Our model failed T3 and T5. Note that option display is difficult to achieve using a memory network, as a memory network cannot recognize "1" and "2" as different numbers. We did not find a method of overcoming this number recognition problem. Further, although we used meta data type features with forms such as <R_cuisine.1>, which resolves the OOV problem well, it was difficult for the model to recognize the difference between <R_cuisine.1> and <R_cuisine.2>.

A model employing two output layers, meta data type features, and an OOV Change feature was also examined (Table 5). Here, the OOV Change feature is a rule-based OOV change function that changes OOV words that do not exist in the KB. Through this method, an OOV is converted to meta data type. For comparison, the results output by the basic two-output-layer model are also listed in Table 6. As the model with meta data type features only (Table 4) exhibited the best performance.

Table 4: *Precision results for top-1 answers in each test set (Tst) for D_layer model with meta data type features.*

Task	Tst 1	Tst 2	Tst 3	Tst 4
1	0.929	0.929	0.645	0.679
2	0.994	0.991	0.649	0.646
3	0.744	0.734	0.719	0.717
4	1.0	1.0	1.0	1.0
5	0.784	0.795	0.681	0.713

Table 5: *Precision results for top-1 answers for D_layer model with meta data type features and OOV Change.*

Task	Tst 1	Tst 2	Tst 3	Tst 4
1	0.934	0.927	0.618	0.672
2	0.995	0.994	0.634	0.634
3	0.73	0.724	0.726	0.714
4	1.0	1.0	1.0	1.0
5	0.751	0.775	0.661	0.663

Table 6: *Precision results for top-1 answers for basic D_layer model.*

Task	Tst 1	Tst 2	Tst 3	Tst 4
1	0.746	0.753	0.62	0.614
2	0.656	0.628	0.645	0.639
3	0.596	0.603	0.615	0.575
4	0.548	0.566	0.645	0.573
5	0.673	0.66	0.688	0.65

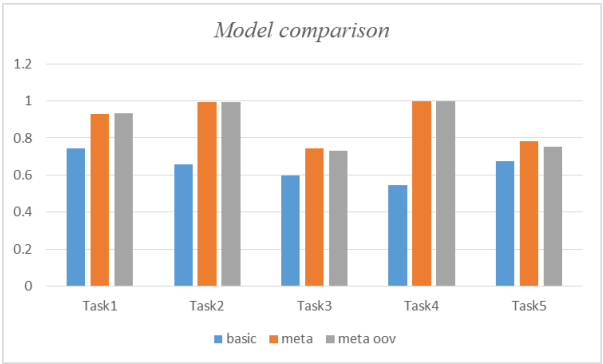

Figure 7: *Model comparison for Tst 1 task scores (basic: basic two-output-layer model; meta: basic with meta data type features; meta oov: meta with OOV Change).*

Fig. 7 shows a comparison of the model performance for the different tasks and Tst 1. We expected that the model incorporating meta data type features and the OOV Change technique (Table 5; "meta oov" in Fig. 7) would yield the best score. However, the model featuring meta data type features only (Table 4; "meta" in Fig. 7) exhibited the best performance. The change rule implemented in OOV Change may be problematic. Therefore, identification of this problem will be the focus of our future work.

Finally, we tested a model incorporating a word feature layer. We experimented with this model after the competition and, thus, it was not submitted to the competition. The results are listed in Table 7. The precision increased to 0.987, 0.999, and 0.964 for T1, T3, and T5, compared to 0.981, 0.997, and 0.922, respectively, for the model with meta data type features (Table 3). The precision was the same for both models for T4. However, the precision was slightly lower for the model with the word feature layer (0.992) compared to the model with meta data type features only (0.995) for T2.

Table 7: *Average precision of top-1 answers of final D_layer model with meta data type features and word-feature linear layer.*

Task	D_layer model + word feature linear layer + meta data type features	D_layer model + meta data type features
1	0.987	0.981
2	0.992	0.995
3	0.999	0.997
4	1.0	1.0
5	0.964	0.922

5. Conclusions

We attempted to develop a model that effectively performs the tasks of DSTC6 Track 1, i.e., end-to-end goal-oriented dialog learning that finds the correct answer in sentence form for a question in dialog. We found that memory networks using meta data type features yield high performance for the challenge tasks. Further, the addition of a word feature layer yields superior performance. However, no improvement is obtained using an additional output representation layer (the D_layer). Additional experiments were conducted with the word feature layer model using a pretrained embedding vector; however, good performance was not achieved. In future research, we will apply this model to other language tasks.

6. Acknowledgements

This material is based upon work supported by the Ministry of Trade, Industry & Energy (MOTIE, Korea) under Industrial Technology Innovation Program (No.10063424, 'development of distant speech recognition and multi-task dialog processing technologies for in-door conversational robots').

This work was supported by the ICT R&D program of MSIP/IITP. [R0126-16-1112, Development of Media Application Framework based on Multi-modality which enables Personal Media Reconstruction].

7. References

[1] Dialog System Technology Challenges. http://workshop.colips.org/dstc6/index.html

[2] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merriënboer, Armand Joulin, Tomas Mikolov, "Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks", *arXiv: 1502.05698, 2015*

[3] Antoine Bordes, Y-Lan Boureau, Jason Weston, "Learning End-to-End Goal-Oriented Dialog", *arXiv: 1605.07683, 2017*

[4] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, Rob Fergus, "End-To-End Memory Networks", *arXiv: 1503.08895, 2015*

[5] J. Weston, S. Chopra, A. Bordes. "Memory Networks." In International Conference on Learning Representations (ICLR), 2015

[6] Tensorflow. https://www.tensorflow.org/

[7] Jeffrey Pennington, Richard Socher, Christopher D. Manning, "GloVe: Global Vectors for Word Representation", Empirical Methods in Natural Language Processing (EMNLP), 2014