An Adaptive Response Matching Network for Ranking Multi-turn Chatbot Responses

Disen Wang[1(✉)] and Hui Fang[1,2]

[1] Institute for Financial Services Analytics, University of Delaware, Newark, USA
{disen,hfang}@udel.edu
[2] Department of Electrical and Computer Engineering, University of Delaware, Newark, USA

Abstract. With the increasing popularity of personal assistant systems, it is crucial to build a chatbot that can communicate with humans and assist them to complete different tasks. A fundamental problem that any chatbots need to address is how to rank candidate responses based on previous utterances in a multi-turn conversation. A previous utterance could be either a past input from the user or a past response from the chatbot. Intuitively, a correct response needs to match well with both past responses and past inputs, but in a different way. Moreover, the matching process should depend on not only the content of the utterances but also domain knowledge. Although various models have been proposed for response matching, few of them studied how to adapt the matching mechanism to utterance types and domain knowledge. To address this limitation, this paper proposes an adaptive response matching network (ARM) to better model the matching relationship in multi-turn conversations. Specifically, the ARM model has separate response matching encoders to adapt to different matching patterns required by different utterance types. It also has a knowledge embedding component to inject domain-specific knowledge in the matching process. Experiments over two public data sets show that the proposed ARM model can significantly outperform the state of the art methods with much fewer parameters.

Keywords: Response selection · Multi-turn chatbot · Response ranking

1 Introduction

With the prevalence of intelligent personal assistant systems, it becomes increasingly important to build an effective chatbot that can communicate with humans fluently and help them fulfill different tasks. Moreover, chatbots also play an important role in many other applications such as customer support and tutoring systems [1,10,11].

One of the key components of any chatbots is the underlying response matching model, whose goal is to identify the most appropriate response from a pool

E. Métais et al. (Eds.): NLDB 2020, LNCS 12089, pp. 239–251, 2020.
https://doi.org/10.1007/978-3-030-51310-8_22

of candidates given past conversations. The past conversations include both user inputs and the system's previous responses. As illustrated in Fig. 1, a user is having a multi-turn conversations with the system (e.g., chatbot). Given the current input "something is slowing me down bad" as well as the past utterances, the system is expected to identify the most appropriate candidate response, i.e., "what is the process name with the highest cpu usage?", among all the candidates.

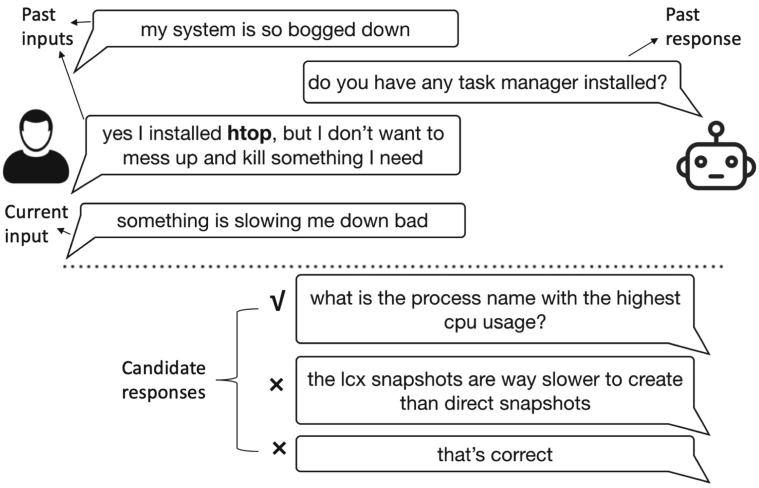

Fig. 1. An example conversation

An optimal response matching model needs to address the following three challenges. (1) It needs to capture a matching relationship that goes beyond simple lexical or semantic similarities. As shown in the Fig. 1, the current input and its correct response do not share any similar words, so the models that are only based on word similarity would not work well for this problem [5,6,17,19]. (2) It should be able to capture different matching relationships for different utterance types. Past utterances include two types of information: past inputs from the user, and past responses from the system. Intuitively, given a candidate response, its desirable matching relationship with past inputs should be different from that with past responses. For example, the correct response needs to address the questions or concerns described in the current input, while it needs to avoid repeating the same information as those mentioned in the past responses. (3) It can utilize domain knowledge to understand the specific meanings of some utterances. For example, "htop" is a process monitoring application for Linux. Without understanding the meaning of this command, it would be difficult for the system to figure out that htop is related to the cpu usage.

Most existing studies on multi-turn response selection [8,16–18,20] mainly focused on the first challenge (i.e., modeling the matching relationships that go beyond semantic similarity), but little attention has been paid to address the last two.

In this paper, we propose an adaptive response matching network (ARM) to address all three challenges. Specifically, a novel response matching encoder is proposed to model the response matching relationship between each utterance and response pair. For each utterance-response pair, instead of computing the matching score based on word or segment level, the encoder computes a response matching matrix through transfer matrices and the multi-attention mechanism. To adapt to the different utterance types, separate encoders are trained: one encoder is used to capture the relationship between current response and the past responses, while the other is used to model the relationship between the current response and the past input. Finally, a knowledge embedding layer is added to the model, and such a layer enables the model to leverage domain knowledge during the response matching process. To validate the proposed ARM model, we conduct experiments over two public data sets. Experimental results and further analysis demonstrate the proposed ARM model is able to achieve superior performance in terms of both effectiveness and efficiency when compared with the state of the art methods.

2 Related Work

Given the past conversations, a chatbot can either automatically generate responses [7,12,15], or retrieve the most appropriate response from a pool of candidates [8,17,19,20]. This paper focuses on the retrieval-based models for multi-turn response selection, and we now briefly review the related work in this area.

Early studies concatenated all of the past utterances together and computed the matching score between the merged utterance and each candidate response [5,6,8]. Specifically, the **Dual-encoder** model [8] used two LSTMs to generate the embeddings for the utterances and candidate response respectively to compute the matching score. The deep relevance matching model (**DRMM**) [5] and the **ARC-II** model [6] were proposed for ad-hoc retrieval tasks, but they were also applied to the response selection problem when past conversations were used as queries and candidate responses were used as documents.

One limitation of these models is that they concatenated all the previous utterances before embedding. Sequential matching network (**SMN**) [17] was proposed to address this limitation. It treated each utterance separately, computed the word-level and segment-level similarity for each response-utterance pair, and used a CNN to distill matching features. Deep matching network (**DMN**) [19] also treated each past utterance separately. It extracted relevant question-answer posts from the collection and utilize the pseudo relevance feedback methods to expand candidate responses with the extracted Q/A pairs. These methods are mainly based on the word semantic similarity scores [2,17]. However, semantic similarity is not sufficient to capture the relationship between a past utterance and a candidate response as shown in Fig. 1.

More recently, inspired by the Transformer structure [14], the attention mechanism has been applied to find better semantic representations of the utterances. Deep attention model (**DAM**) [20] proposed self-attention and cross-attention

to construct semantic representations at different granularity, and the multi-representation fusion network (**MRFN**) [13] has further applied multiple representation strategies for utterances and fused them in the final step to compute the matching scores.

All these previous studies mainly tackled the first challenge discussed in Sect. 1, and focused on developing models to capture the matching relationship between the past utterances and candidate responses. Few of them studied how to adapt the matching model to different types of utterances and how to incorporate the domain knowledge in a more general way, which is the focus of our paper.

3 Adaptive Response Matching Network

3.1 Problem Formulation

The problem of multi-turn response selection can be described as follows. Assume we have a training data set $\mathcal{D} = \{(u_i, r_i, l_i)\}_{i=1}^{N}$, where $u_i = \{p_{i,1}, q_{i,2}, ..., p_{i,n_i}\}$ denotes a conversation consisting of multiple utterances. Each utterance could be either a previous input ($p_{i,j}$) or a previous response ($q_{i,j}$). r_i denotes a candidate response for u_i, and $l_i \in \{0, 1\}$ indicates whether r_i is the correct response for u_i. The task is to learn a model f based on $\mathcal{D}$, which can compute the response matching score between any response r and utterance u. Given a new utterance, the learned model f will rank a list of candidate responses based on their matching scores, and the one with the highest score will be selected as the final response.

In this paper, we will also study how to incorporate the domain knowledge in the training process. Specifically, we assume that the knowledge base can be denoted as follows: $\mathcal{K} = \{(c_i, g_i)\}_{i=1}^{M}$, where c_i is a domain-specific keyword, and g_i is the corresponding description of that keyword. Take the conversations shown in Fig. 1 as an example, c_i could represent the command *htop* and g_i would be the description of that command. Both $\mathcal{D}$ and $\mathcal{K}$ will be used to learn the model f.

3.2 Overview of the ARM Model

The key challenge in multi-turn response selection model lies in how to model the matching relationship between a candidate response and the past utterances. Almost all of the recent studies followed a three-step procedure: representation, matching and accumulation [2,17]. The first step is to represent candidate responses and utterances in various ways that can capture their semantic meanings. The second step is to compute the matching scores between the utterances and the candidate response based on these representations. And the last step is to accumulate all the scores into the final one.

Our proposed adaptive response matching network model (ARM) also follows the above three-step procedure, with major differences in the first two steps. Figure 2 shows the overall architecture of the ARM model. Compared with the existing studies, ARM aims to develop a matching mechanism that can adapt

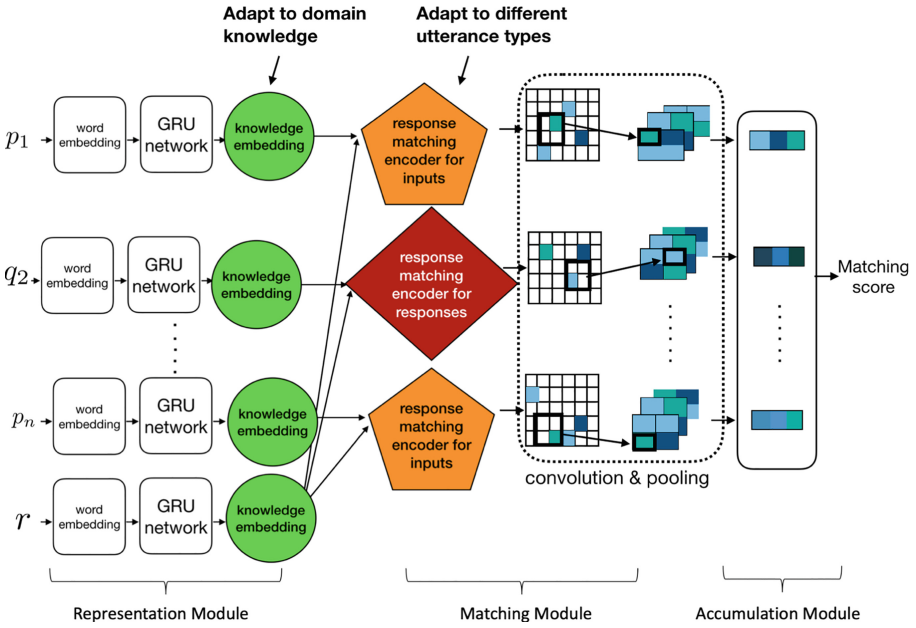

Fig. 2. Overview of the adaptive response matching network (ARM)

to the domain knowledge (in the representation step) and different types of utterances (in the matching step).

In the ARM model, the *representation module* first converts each utterance into a segment-level vector representation using word embedding and GRU, and then enhances the vector representation with domain knowledge using knowledge embedding layer. After that, *the matching module* utilizes the representations of the utterance vectors and the response vectors to calculate the response matching score, and extracts features to feed into the accumulation module. More specifically, multiple transfer matrices are trained to transfer hidden states vectors into different representations spaces, and the combination of response matching matrix from each representation space is used to compute the matching score in the response matching encoder. The segments in an utterance that are important for recognizing appropriate response will have higher matching scores. The areas with higher scores in the final relevance matrix will be extracted by CNN network. Finally, the *accumulation module* generates the final matching score based on the matching vectors provided by the matching module.

3.3 Adaptive Response Matching Encoder

A past utterance could be either an input from the user or a response from the system. Both utterance types are useful to select the matching candidate response, but in a different way. An input often describes a request or a problem encountered by a user, so the correct candidate response is expected to be

the solution to the inputs. On the contrary, a candidate response is expected to be a follow-up or clarification of previous responses. Let us take a look at the example illustrated in Fig. 1 again. The correct response, i.e., "the process name with the highest cpu usage", is expected to be the solution to the request related to "something is slowing me down". On the other hand, it is a follow-up question to the previous response, i.e,. "do you have any task manager installed?" Clearly, it is necessary to ensure the matching model be adaptive to different utterance types. In other words, when selecting a response for the given utterances, different matching mechanisms need to be used for past inputs and past responses.

Although a few models have been proposed to solve the problem of multi-turn response selection [2,13,17,20], none of them studied how to directly adapt the matching mechanisms to different utterance types. Instead, they mainly focused on exploring various complicated representations of the utterances with the hope that these representations can better capture the semantic meanings of the utterances. Although the more complex representations can lead to better effectiveness, they often require more computational resources and longer time to train and test.

In the ARM model, we propose adaptive response matching encoders to learn different matching patterns according to the utterance types. The basic idea of ARM encoders is to start with some basic semantic representations of the utterances/responses, and then learn new matching representations for each matching type. The new matching representations are expected to better capture the response matching relationship for each utterance type. We now describe the encoders in detail, and the important notations are summarized in Table 1.

Starting with Basic Representations: Following the previous study [17], we represent both responses (r) and utterances (either p for a previous user input or q for a previous response) using segment-level representation. Specifically, we first apply Word2Vec [9] algorithm to generate the word embedding $e \in \mathbb{R}^d$ for each word, where d is the number of dimensions in the word embedding. The model looks up a pre-trained word embedding table to convert $p = [w_{p,1}, w_{p,2}, ..., w_{p,n_p}]$ into $P = [e_{p,1}, e_{p,2}, ..., e_{p,n_p}]$, where $w_{p,i}$ is the i-th word and $e_{p,i}$ is the corresponding word embedding vector, and n_p is the length of p. Similarly, we can represent q as $Q = [e_{q,1}, e_{q,2}, ..., e_{q,n_q}]$ and r as $R = [e_{r,1}, e_{r,2}, ..., e_{r,n_r}]$ where $e_{q,i}, e_{r,i} \in \mathbb{R}^d$, are the embeddings of the i-th word of q and r, and n_q and n_r are the length of q and r. To extract the contextual information in each utterance, we feed P, Q and R into GRU network [3] and use the generated hidden states H_p, H_q and H_r as basic representations of the responses and utterances.

Learning New Matching Representations: Given the basic representations of a previous utterance and a response (e.g., H_p and H_r), two transfer matrices (i.e., W_p and W_r) are learned to transfer the basic representations to the new ones (i.e., RM_p and RM_r) that can better capture the response matching relationship. Formally, the new representations (i.e,. relevance matrix) of the utterance and the response can be computed using

Table 1. Explanations of key notations.

Notation	Explanation
p	a past input
q	a past response
u	an utterance, could be either p or q
r	a candidate response
w_i	a word
e_i	the word embedding vector of word w_i
h_i	the hidden state vector generated by GRU for word w_i
h_i^k	the basic knowledge embedding vector of w_i
h_i^{new}	the new knowledge embedding vector of w_i after the gating mechanism
H	hidden state matrix of a utterance or response (based on h_i or h_i^{new})
W	transfer matrix in the adaptive response matching encoder
RM	relevance matrix in the adaptive response matching encoder
M	single-head response matching matrix
M'	final response matching matrix in multi-head encoder
W^s, W^g	learning parameters for the knowledge embedding layer

$$RM_p = H_p W_p, RM_r = H_r W_r$$

where $W_p, W_r \in \mathbb{R}^{h \times m}$, h is the number of dimensions in hidden states, and m is the number of dimensions in the response matching encoder. W_p and W_r are transfer matrices and will be learned from the training data. The initial weights of these matrices are randomly initialized with different values, and the training data contain the labels indicating whether a utterance is a input or a response.

Computing Matching Scores: With the newly learned matching representations of a utterance and a candidate response, we can compute a response matching matrix as follows.

$$M = softmax(\frac{RM_p {RM_r}^T}{\sqrt{m}}).$$

Specifically, $M[i, j]$ represents the response matching score of the i-th hidden state from H_p and the j-th hidden state from H_r. The i-th hidden state from the utterance (i.e., $h_{p,i}$) is first transferred to a vector in a new representation space (i.e., RM_{pi}) through the transfer matrix (i.e., W_p). Similarly, the j-th hidden state from the response (i.e., $h_{r,j}$) is converted to another vector in the new representation space (i.e., RM_{rj}) through W_r. The response matching score of these two new vectors is then computed using $M_{i,j} = softmax(\frac{RM_{pi} \cdot RM_{rj}^T}{\sqrt{m}})$.

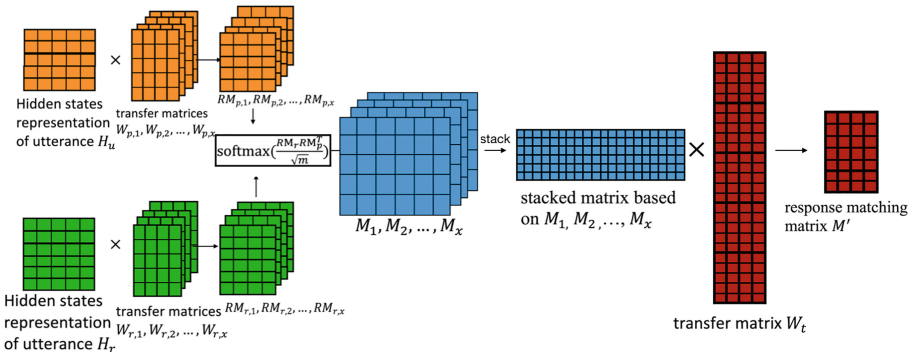

Fig. 3. Details of adaptive response matching encoders

Improvement via Multi-head Mechanism: In order to capture different representations that could be useful for response matching, we apply the multi-head attention mechanism [14] to the response matching encoder. The structure of multi-head encoder is shown in Fig. 4. We learn different transfer matrices to transfer the hidden states vectors into different representation spaces, and combine the matrices from each representation space to get the final matrix M'. Specifically, we stack matrices $M_1, M_2, ..., M_x$ into one matrix and multiply it with a transfer matrix W_t to get the final response matching matrix M', where the elements of W_t are learning parameters. W_t learns how to combine information from multiple channels into the final response matching matrix. In the final response matching matrix M', elements with higher values mean the corresponding word pairs have higher semantic matching scores. This multi-head mechanism can expand the model's ability to focus on different positions, and it is able to capture more diverse response matching patterns through the multiple representation spaces (Fig. 3).

3.4 Knowledge Embedding Layer

An optimal response matching model also needs to adapt to the domain-specific knowledge. As explained in Fig. 1, if we have domain-specific knowledge about "htop", we might be able to better match the candidate responses. Assume the available knowledge base can be represented as pairs of domain-specific concepts and their descriptions (i.e., (c_i, g_i)), we now discuss how our model adapt to domain knowledge.

A straightforward way to incorporate knowledge base is to replace the domain-specific words with their definitions or descriptions in the corresponding hidden states. However, words are ambiguous, and not every occurrence of a domain-specific word refers to the same meaning. For example, the word "install" is a command in the Ubuntu system. But if a user asks "how to install the task manager", the word "install" in this utterance should not be replaced by the description of the command because it did not refer to the command. To tackle this problem, we propose to apply gating mechanisms to decide when and how

to fuse knowledge into utterance representations in the knowledge embedding layer [4].

For each word w_i that occurs in either the utterances or the response, if it is one of the domain-specific word (c_i), we would extract its corresponding description from the knowledge base (i.e., g_i), feed g_i into a GRU unit, and the generated hidden state is used as knowledge embedding, which is denoted as h_i^k. After that, we can apply gating mechanism to fuse knowledge embedding h_i^k into word representation h_i to generate the new representation h_i^{new}:

$$h_i^{new} = bh_i + (1-b)\tanh W^s[h_i; h_i^k; h_i - h_i^k]$$

$$b = sigmoid(W^g[h_i; h_i^k; h_i - h_i^k])$$

where W^s and W^g are learning parameters, and $[v_1, v_2, v_3]$ means to stack the three vectors into a matrix.

With the gating mechanism, the new word representation h_i^{new} has selectively adapt to the domain knowledge based on the context information. For each dimension in the vector h_i, the gating mechanism decides whether to keep the original value or replace it with the value from knowledge embedding. Without the gating mechanism, the model would replace values in all dimensions with those from knowledge embedding, which might not be always the best solution. The new representation h_i^{new} is then used to replace h_i in the hidden state matrix H_p, H_q and H_r, i.e., the input of the adaptive response matching encoders.

3.5 Summary

The ARM model can be regarded as an extension of the SMN model [17] with a couple of notable differences. First, the model uses adaptive response matching encoders to learn different matching patterns according to the utterance type, and the captured matching relationship is able to go beyond the simple semantic similarity. Second, the model adds a knowledge embedding layer (in the representation module) to provide a general way to incorporate domain knowledge. These two differences enable the ARM model better capture the response matching relationship, and explain why the ARM model is more effective than the state of the art models.

DMN [19] is the only existing study that tried to utilize knowledge base. It extracted question-answer pairs from the collection as knowledge, and then utilized the pseudo relevance feedback methods to expand candidate responses with the extracted knowledge. On the contrary, the ARM model presents a more general and robust way of incorporating domain-specific knowledge. In particular, the representation of the knowledge base is more general. The knowledge can come from either collection itself or external domain-specific resources. Moreover, the gating mechanism in the embedding layer makes it possible to selectively apply the domain knowledge based on the context information, which can improve the robustness of the model.

Another major advantage of ARM model lies in its efficiency. When designing the model, we intentionally use the basic semantic representations (i.e.,

word/knowledge embedding and GRU outputs) and let the transfer matrices to learn useful matching relationships for each utterance type. Such a learning process is more targeted than learning a general yet more complicated representation that could be useful to match all kinds of utterance types [13,20]. As shown in Sect. 4.2, our model uses much fewer parameters than the state of the art models, yet it is able to achieve better performance.

4 Experiments

4.1 Experiment Setup

Data Sets: We evaluate the performance of the proposed model on the two publicly available data sets from the DSTC7 Response Selection Challenge[1]. (1) The first data set is the *Ubuntu dialogue corpus*, which contains conversations about solving an Ubuntu user's posted problem. The domain knowledge includes the commands and their corresponding function descriptions. The training set contains 1 million conversations, the development set and the test set each contains 0.5 million conversations. (2) The second data set is the *student-advisor data set*. In each conversation, the advisor will guide the student to pick courses. The external knowledge is about the courses such as their descriptions and areas. The training set contains 0.5 million conversations, development set and test set each contains 50,000 conversations.

Evaluation Measures: Following the previous studies [8,17], the primary evaluation measures are $R_{100}@k$, which is the recall at position k in 100 candidates. k is set to 1, 5, 10. Since there is only one correct response for each conversation, the precision at position k always equal to the recall at position k divided by k. Thus, only the values of recall are reported.

Table 2. Performance comparison. * means statistically significant difference over the best baseline with $p < 0.05$ under student's t-test.

	Ubuntu data set			Student-advisor data set		
Model	$R_{100}@1$	$R_{100}@5$	$R_{100}@10$	$R_{100}@1$	$R_{100}@5$	$R_{100}@10$
Dual-encoder	18.3%	34.35%	47.15%	11.35%	19.08%	33.2%
DRMM	21.25%	37.83%	52.48%	15.41%	24.92%	36.75%
ARC-II	20.31%	36.53%	49.42%	14.85%	23.42%	37.25%
SMN	34.14%	59.13%	71.52%	19.57%	28.2%	52.39%
DMN	33.91%	58.9%	70.86%	19.6%	38.23%	51.64%
DAM	35.37%	61.18%	72.29%	21.13%	40.57%	55.32%
MRFN	36.13%	63.45%	77.85%	20.35%	38.92%	54.28%
ARM	**39.93%***	**67.21%***	**78.95%***	**23.74%***	**41.83%***	**58.5%***

[1] https://github.com/IBM/dstc7-noesis.

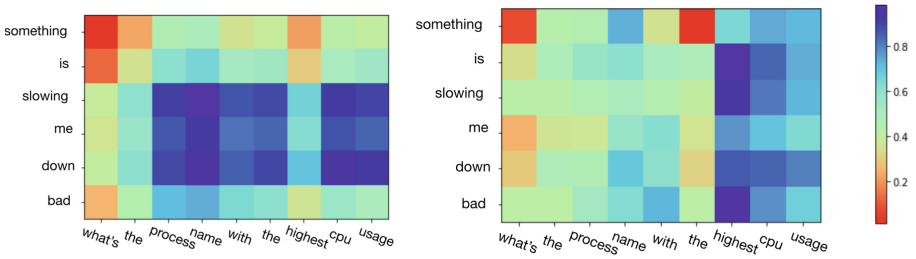

Fig. 4. Response matching matrix in ARM (left) vs. similarity matrix in SMN (right)

Table 3. Ablation analysis results

	Ubuntu data set			Student-advisor data set		
Model	R_{100}@1	R_{100}@5	R_{100}@10	R_{100}@1	R_{100}@5	R_{100}@10
ARM-K	36.74%	63.93%	75.12%	22.38%	38.96%	55.82%
ARM-S	38.35%	65.05%	75.32%	23.27%	40.15%	55.26%
ARM	**39.93%***	**67.21%***	**78.95%***	**23.74%***	**41.83%***	**58.5%***

Parameter Settings: In our model, dropout layer was added to the CNN network, and the dropout rate is set to 0.5. Zero padding was applied to make the size of interaction matrix same for all utterance-response pairs. The size of interaction matrix after padding is 50×50. We also set the maximum length of utterance and response as 50 words. The learning rate was set as 0.0005, and experiments show that larger learning rate would lead to a significant decrease in performance. We applied Word2Vec algorithm to train the word embedding matrix, and the number of word embedding dimensions is 200. All these hyper-parameters are chosen to be consistent with the previous study [17].

4.2 Performance Comparison

We compare the performance of the proposed ARM model with several state of the art baseline methods over both data sets. In particular, the baseline models include the following seven state of the art response matching models: Dual-encoder [8], DRMM [5], ARC-II [6], SMN [17], DMN [19], DAM [20] and MRFN [13], which were briefly reviewed in Sect. 2. The implementations of the baseline models were obtained either through the code published by the authors or MatchZoo[2] on the Github.

The results of performance comparison are summarized in Table 2. It is clear that our proposed ARM model outperforms all the strong baseline models significantly on both data sets, which indicates the effectiveness of the ARM model in addressing the limitations we discussed before. We conduct additional experiments to better understand the proposed model.

The proposed ARM model is an extension of the SMN model [17] with the addition of two important components: adaptive response encoders and the

[2] https://github.com/NTMC-Community/MatchZoo.

knowledge embedding layer. To better understand the effectiveness of these additions, we conduct additional experiments by disabling each of these new additions, and the results are shown in Table 3. ARM-S is the variation of ARM model, where we disable the separate encoders and use the same response encoders for different utterance types. ARM-K is the variation of ARM, where we disable the knowledge embedding layer. When we compare the two variations with the ARM model, it is clear that both type-adapted encoders and the knowledge embedding layer are useful to improve the performance. Another key difference between ARM and SMN model is how to model the response matching relationships. SMN mainly relies on the semantic similarity, while the proposed response matching encoder can capture other semantic matching relationship. The difference can be easily seen through an example shown in Fig. 4. The darker color means a higher score. It is clear that the ARM model can correctly capture the matching relationship between {*slowing, me, down*} from the input and {*process, name, with, the, cpu, usage*} from the correct response, while the SMN cannot.

Furthermore, ARM is shown to be more effective than the recently proposed DAM and MRFN models according to Table 2. In fact, it is also more efficient. Similar to DAM and MRFN, ARM aims to capture the matching relationship that goes beyond simple semantic similarity. Instead of learning general yet more complicated representations for all utterances, ARM is more focused and it explicitly adapts to different matching patterns caused by different utterance types. As a result, ARM is more efficient and requires much fewer parameters than DAM and MRFN. With the same hyper-parameter values (e.g., the number of dimensions in Word2Vec vectors), the numbers of parameters used in the ARM, DAM and MRFN models are 30 millions, 74 millions, and 91 millions respectively. Fewer parameters means significant improvement in terms of efficiency.

5 Conclusions and Future Work

In this paper, we propose an adaptive response matching network for multi-turn chatbot response selection. Existing models focused on modeling different relationships using multiple representation strategies, while the ARM utilizes adaptive response matching encoders in the matching module to directly model different matching relationships for different types of utterances. Moreover, ARM has a knowledge embedding layer, which can adapt to external domain knowledge in a general way. Empirical results over two data sets show that the proposed model outperforms various state-of-the-art models in terms of both effectiveness and efficiency. In the future, we plan to study how to incorporate unstructured domain knowledge to further improve performance.

Acknowledgement. The first author is grateful to the JP Morgan Chase scholarship he received from the Ph.D. Program in Financial Services Analytics to support this research.

References

1. Assefi, M., Liu, G., Wittie, M.P., Izurieta, C.: An experimental evaluation of Apple Siri and Google Speech Recognition. In: ISCA SEDE, pp. 1–6 (2015)
2. Chen, Q., Wei, S., Inkpen, D.: Enhanced LSTM for natural language inference. In: ACL, pp. 1657–1668 (2017)
3. Cho, K., et al.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: EMNLP, pp. 1724–1734 (2014)
4. Gers, F., Schmidhuber, J., Cummins, F.: Learning to forget: continual prediction with LSTM. Neural Comput. **12**(1), 2451–2471 (2000)
5. Guo, J., Fan, Y., Ai, Q., Croft, W.B.: A deep relevance matching model for ad-hoc retrieval. In: CIKM, pp. 55–64 (2017)
6. Hu, B., Lu, Z., Li, H., Chen, Q.: Convolutional neural network architectures for matching natural language sentences. Adv. Neural Inf. Process. Syst. **3**(1), 2042–2050 (2015)
7. Li, J., Galley, M., Brockett, C., Spithourakis, G.P., Gao, J., Dolan, B.: A persona-based neural conversation model. In: ACL, pp. 994–1003 (2016)
8. Lowe, R., Pow, N., Vlad, I., Charlin, L., Liu, C.W., Pineau, J.: Training end-to-end dialogue systems with the Ubuntu Dialogue Corpus. Dialogue Discourse **8**(1), 31–65 (2017)
9. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In: ICLR, pp. 1–12 (2013)
10. Qiu, M., et al.: AliMe chat: a sequence to sequence and rerank based chatbot engine. In: ACL, pp. 498–503 (2017)
11. Shum, H.Y., He, X., Li, D.: From Eliza to XiaoIce: challenges and opportunities with social chatbots. Front. Inf. Technol. Electron. Eng. **19**(1), 10–26 (2018). https://doi.org/10.1631/FITEE.1700826
12. Sordoni, A., et al.: A Neural Network Approach to Context-Sensitive Generation of Conversational Responses. arXiv e-prints arXiv:1506.06714 (2015)
13. Tao, C., Wu, W., Xu, C., Hu, W., Zhao, D., Yan, R.: Multi-reprentation fusion network for multi-turn response selection in retrieval-based chatbots. In: WSDM, pp. 267–275 (2019)
14. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
15. Wen, T.H., et al.: A network-based end-to-end trainable task-oriented dialogue system. In: ACL, pp. 438–449 (2017)
16. Wu, B., Wang, B., Xue, H.: Ranking responses oriented to conversational relevance in chat-bots. In: COLING, pp. 652–662 (2016)
17. Wu, Y., Wu, W., Xing, C., Xu, C., Li, Z., Zhou, M.: A sequential matching framework for multi-turn response selection in retrieval-based chatbots. In: ACL, pp. 496–505 (2017)
18. Yan, R., Song, Y., Wu, H.: Learning to respond with deep neural networks for retrieval-based human-computer conversation system. In: SIGIR, pp. 55–64 (2016)
19. Yang, L., Huang, J., Chen, H., Croft, W.B.: Response ranking with deep matching networks and external knowledge in information-seeking conversation systems. In: SIGIR, pp. 245–254 (2018)
20. Zhou, X., et al.: Multi-turn response selection for chatbots with deep attention matching network. In: ACL, pp. 1118–1127 (2018)