A PIPELINE-BASED TASK-ORIENTED DIALOGUE SYSTEM

ON DSTC2 DATASET

by

Yize Pang

APPROVED BY SUPERVISORY COMMITTEE:

Vincent Ng, Chair

Latifur Khan

Sriraam Natarajan

This thesis is dedicated

to my parents Bo Pang and Yanqun Zheng,

to my friends, for the support and love they give me.

A PIPELINE-BASED TASK-ORIENTED DIALOGUE SYSTEM ON DSTC2 DATASET

by

YIZE PANG, BE

THESIS

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

May 2019

ACKNOWLEDGMENTS

I would first like to thank my advisor Dr. Vincent Ng for the chance to work with him and for the valuable guidance in the past year. I would also like to thank Dr. Latifur Khan and Dr. Sriraam Natarajan for the help they gave me and serving as my thesis committee.

I would like to thank my friends Mingqing Ye, Jing Lv, Zixuan Ke, Hui Lin, for the discussions, suggestions and a variety of matters during the last semester.

Finally, I must express gratitude to my parents for the continuous support and encouragement throughout my life.

April 2019

A PIPELINE-BASED TASK-ORIENTED DIALOGUE SYSTEM

ON DSTC2 DATASET

Yize Pang, MSCS
The University of Texas at Dallas, 2019

Supervising Professor: Vincent Ng, Chair

Dialogue systems have attracted a lot of attention since some conversational products like Google Assistant and Amazon Echo smart speaker have achieved big successes recently. In this work, we try to build a pipeline-based task-oriented dialogue system, which is the core technology behind these famous products. Our system consists of three modules: a GLAD dialogue state tracker, a policy learning module and a response generation module. They are sequentially connected. The contributions of this work are two-fold. Firstly, we propose an effective approach to improve the controllability of language generation. The experimental results show that this strategy significantly increases the key information accuracy in the generated dialogue responses. Secondly, we introduce a practical method to build a task-oriented dialogue system. Compared to models that are completely based on neural networks, the modularity of our system helps convert a hard problem into several smaller ones that are more specific and easier to solve.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Dialogue system is an attractive problem in the field of natural language processing. Since the invention of the computer, building the machine that can communicate with humans has always been a goal pursued by mankind. Recently with the development of deep learning, the dialogue systems we can build are becoming powerful. Actually, with the help of automatic speech recognition, technology giants have built some products in the field of personal assistant and intelligent furniture by using the technique of dialogue system. For example, Google Assistant and Amazon Alexa are two famous applications that are popular in common life. And accompanied by the success of these products, research in the dialogue system has also become hot in recent years.

According to the definition in the book of Speech and Language Processing (Keselj, 2009), a dialogue system (also known as conversational agent) is a kind of program that can communicate with human in natural language and it can be roughly divided into two categories: task-oriented dialogue systems and chat bots (which means open-domain dialogue systems).

Task oriented-dialogue systems are only designed for a specific task like booking a flight ticket or restaurant reservation. Usually, their abilities are very limited, however task-oriented dialogue systems have been widely used in industry for a long time. For example, in the field of online shopping, many customer services are processed by robots. Although they can only handle some simple questions, such automation can significantly reduce the labor cost for a company.

Chat bots are designed for communication in open topic. Theoretically, you can talk anything with them. Comparing to the task-oriented dialogue systems, chat bots look more attractive in the first glance. Since it is more like the interaction with human and much closer to our definition of intelligence. There are some successful products in the industry. One

good example is XiaoIce (Zhou et al., 2018), a popular social chat bot created by Microsoft. Millions of people chat with it for entertainment purposes. However, building a product-level chat bot is much harder than a task-oriented system. The technique we are using does not allow a program to really understand the language meaning. It can learn the sentence pattern and output fluent responses for a user utterance, but the information contained in the response is usually wrong.

In this thesis, we try to build a pipeline-based task-oriented dialogue system in the field of restaurant reservation by using the DSTC2 dataset (Henderson et al., 2014). The reason we decide to employ a pipeline framework is that: it is the first choice to build a product-level dialogue system in the industry and we would like to solve problems in the actual work scenario. This is a challenging task because we need to build multiple modules to process the user input step by step and each module is hard enough to be taken as an independent task.

One big problem to build a pipeline-based dialogue system is lack of data. Although there are many task-oriented dialogue corpora, few of them contain all the information that is needed to build a complete dialogue system by using the pipeline method. We choose DSTC2 dataset since it contains all the necessary information (like dialogue states, system actions) that are needed to build a pipeline-based dialogue system.

Data in the DSTC2 dataset comes from a real-world restaurant reservation system. People made phone calls to it, gave some requirements about the restaurant they would like to find (like the food type, price range). The phone calls were transformed into text format by automatic speech recognition (ASR) module, then get processed by the system. DSTC2 dataset contains the text-format log and annotation files for each phone-call. In the data pre-processing, we extract dialogues from these raw files. A dialogue contains multiple turns, and they are chronologically processed by the system. One example dialogue from DSTC2 dataset is shown in the Figure 1.1.

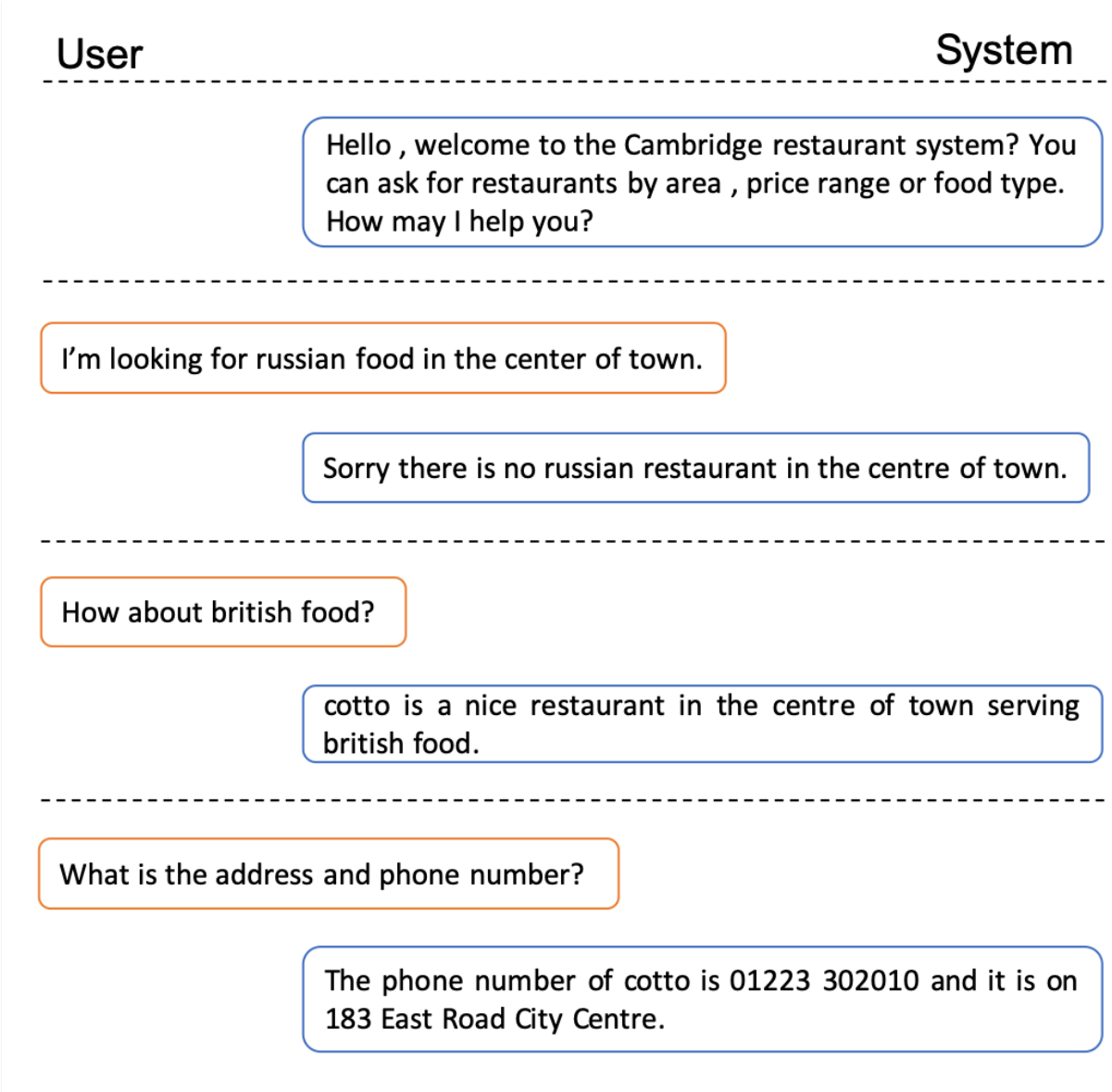


Figure 1.1. An example dialogue from the DSTC2 (Henderson et al., 2014) restaurant reservation dataset. The dialogue contains four turns and is divided by dashed lines. For each turn, the user utterance locates on the left side and it is the input to the dialogue system. The system response is on the right side and it is the output of the system. Both user utterance and system response are in the form of natural language.

Our system consists of three major modules: a GLAD dialogue state tracker, a policy learning module and a response generation module. They are sequentially connected.

Dialogue state tracking is the task to extract formatted information from user utterance and manage the dialogue history. In this way, natural language is transformed into the form that a machine can deal with. We choose the GLAD algorithm (Zhong et al., 2018) to implement the dialogue state tracking module. The reason for this decision is that GLAD is the state of the art on this task. We use it in our system so that the dialogue state tracker will not be the bottleneck to the whole system.

After getting the formatted data, the system needs to query the database to find a suitable restaurant. This work is finished in the policy learning module. What's more, this module needs to decide the actions that the system will take. We use one example to explain the meaning of system actions: query database is an action and the policy learning module is responsible to decide whether or not to trigger this action right now, since the conditions given by user maybe not enough for a query. If the system needs more information from the user, it must return a sentence that informs the user to give more conditions. And the sentence returned to user will be generated under the guidance of some specific system actions (which is predicted in the policy learning module).

Lastly, the response generation module creates a natural language response according to the system actions and restaurant information that comes from the policy learning module. The generation module uses an encoder-decoder architecture (Sutskever et al., 2014). The encoder inputs are system actions, decoder chooses output sentence pattern through it. Besides, the decoder generates a sentence word by word and each one is selected from a vocabulary. There is a big problem for this method: we need to present the restaurant information in the response, but it's hard to precisely control the generation contents. To mitigate this problem, we learn the idea from pointer-generator network (See et al., 2017). That is to modify the output vocabulary distribution and increase the probability of some words according to the restaurant information.

Our contributions contain two parts. The first point is we propose an effective approach to increase the controllability of language generation. The experimental results show that this method significantly increases the accuracy of key information in the generated dialogue responses. Secondly, we build a task-oriented dialogue system on DSTC2 dataset. In this way, we can compare the traditional pipeline-based methods with models that are completely based on neural network, and then we show the pros and cons of each kind of models and finally give the conclusion that the pipeline-based methods are better choices for building task-oriented dialogue systems.

The rest of the thesis is organized as follows. In Chapter 2, we introduce the related work in the field of dialogue systems. In Chapter 3, we explain our model in detail. Chapter 4 gives evaluations and analyses of our system. Lastly, in Chapter 5, we give the conclusion and propose some possible improvements in future work.

CHAPTER 2

RELATED WORK

In this chapter, we present researches in the field of dialogue systems. According to the application areas, dialogue systems can be separated into two groups: task-oriented dialogue systems and chat bots. A task-oriented dialogue system is designed for a particular task like booking flights. Nowadays, most conversation products are based on task-oriented dialogue systems. Some good examples are Amazon Echo smart speaker and voice navigation systems. Another group of dialogue systems is chat bots. The common use for them is to kill time and entertaining.

We firstly introduce the two kinds of systems separately. Then give a review on the evaluation metrics that are used in dialogue systems.

2.1 Task-oriented Dialogue Systems

There are two general methods to build a task-oriented dialogue system. One is the traditional pipeline-based method, the other is the complete neural network based method. Since pipeline methods are the most important ones that are used in task-oriented dialogue systems, we give a detailed introduction to these methods and then briefly introduce the complete neural network based models.

As shown in Figure 2.1, pipeline approaches typically contains four major modules: an NLU module, a dialogue state tracking module, a policy learning module, and an NLG module. They are sequentially connected together.

NLU is the abbreviation of natural language understanding. It is the module that maps natural language user utterances into semantic entity types (known as slots) and determines the user intents. We need to explain the meaning of the term slot and intent. Take weather condition as an example. There are many words that can describe the weather, like sunny,

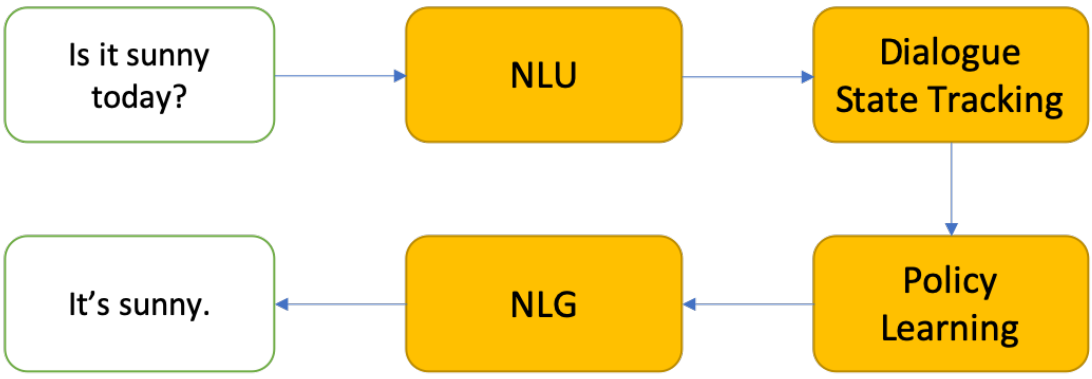


Figure 2.1. Traditional pipeline in task-oriented systems

cloudy. In this case, the weather is a slot, and sunny, cloudy are possible values for this slot. The intent is the purpose that is conveyed in the user utterance. If a user asks "Is it sunny today?", then the user intent is "query weather". Usually, slots and intents are pre-defined in a system. Table 2.1 illustrates the example result of NLG. - refers to no slot.

Table 2.1. Illustration of the example result of NLG.

Sentence	Is	it	sunny	today
Slot	-	-	weather	time
Intent	query weather			

Convolutional neural networks (CNN) are applied to extract query vector representations from user query in search engines (Hashemi et al., 2016). Taking the high-dimensional vectors as a feature, queries are then classified into intent groups. Recurrent neural networks (RNN) are used for slot-filling tasks, and the performances are substantially better than the traditional conditional random fields (CRF) based methods (Mesnil et al., 2013).

Dialogue state tracking is a crucial part of task-oriented dialogue systems. It is used to manage the accumulated and current dialogue states. One dialogue state is a set of user goals

and requests for the current turn. Traditional rule-based methods (Goddeau et al., 1996) and web-style ranking algorithms can be used on dialogue state tracking task (Williams, 2014), however their performances are not good. Recently the boundary between the NLU module and the dialogue state tracking module becomes blurred. Methods like Global-Locally Self-Attentive Dialogue State Tracker (GLAD tracker) (Zhong et al., 2018), Neural Belief Tracker (NBT) (Mrkšić et al., 2016) can finish the intents and slots detection along with the dialogue state tracking. By using previous-turn system actions and comparing all candidate slot-value pairs with utterances, the performance of these neural network based methods obtains great improvements comparing to traditional approaches.

Policy learning module (also known as dialogue management) predicts system actions from dialogue states. A simple transformation by using dialogue states and external database knowledge is used to predict system actions (Wen et al., 2016). Another common approach is the rule-based policy learning (Yan et al., 2017). This approach works for the case that the size of system actions is not very big.

The natural language generation (NLG) component learns the mapping between dialogue actions and natural language responses. Retrieval-based models can be used in an online shopping scenario to generate system responses by comparing the target user query with query-response pairs in a candidate list. After finding the pair whose query is the closest to the target one, the response in this pair is then taken as the system response (Yan et al., 2017). Another widely used models are rule-based or template-based methods (Cheyer and Guzzoni, 2014). This kind of approaches are simple and robustness, but their rigidness and repetition make it hard to generate responses with high diversity. Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Unit (GRU) (Chung et al., 2014) are often used in language generation (Wen et al., 2015). They can learn from unaligned data and easily create sequences with high flexibility.

About the methods that are completely based on neural networks, they don't reduce a dialogue system into smaller components like the pipeline approaches. Instead, they treat

a dialogue system as a whole and usually model it through one end-to-end neural network. The complete neural network methods are more frequently used in chat bots but are less used in task-oriented dialogue systems. In this work, we would like to compare the pipeline methods and complete neural network methods. Thus we choose copy-network (Eric and Manning, 2017) as our main comparison. This model simply uses one end-to-end neural network and we will talk more about it in Chapter 4.

2.2 Chat Bots

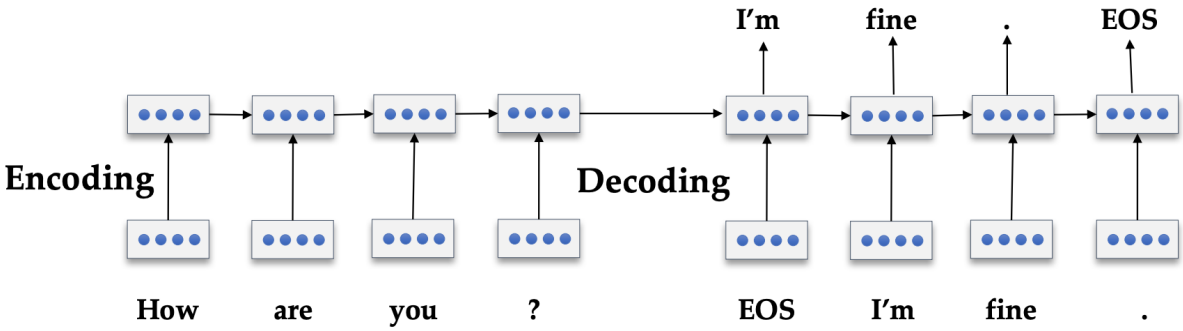
Chat bots are systems that can mimic the unstructured communication between humans. Contrary to the goal-oriented dialogue systems, chat bots are designed to make topic-unlimited conversations with people. Since the huge coverage of contents, recent chat bots are all data-driven. This requires a huge amount of data to build a chat bot. A survey of available corpus (Serban et al., 2015) provides a wealth of useful information about accessible corpora. From this work, we know that available data sources can be social networks like Twitter messaging (Ritter et al., 2010), movie lines (Li et al., 2016), and technology support dialogues (Lowe et al., 2015). These corpora contain a huge amount of data, however, the data is often very noisy and contains a significant number of acronyms, abbreviations that are specific to topics (Clark, 2003). Necessary pre-processing may be needed in such cases (Konstan et al., 1997).

According to the approach used to generate system responses, there are two common methods to build a chat bot, that are retrieval based methods and generation-based methods.

Retrieval-based methods depend on a set of conversation turns. Given a user utterance as query, we compute the similarity (like cosine or inner product) between the query and all turns in the set. After this, we can get the most similar turn to the query. There are two possible returns: the most similar turn or its next turn (the next turn is the response to the current turn). Intuitively the response should be a more reasonable answer. However,

the most similar turn is better in practice (Wang et al., 2013). The information retrieval is basically a matching problem, the only thing that matters is the approach to compute similarity. Deep Neural Networks (DNN) are widely used in this area. (Lu and Li, 2013)

Generation-based methods are sequence transductions from user utterances to system responses. This is firstly introduced by the statistical machine translation on response generation (Ritter et al., 2011). Since the alignment is non-existed between the utterance and response, machine translation performs poorly on this task. However, the encoder-decoder model effectively solves the alignment problem (Sutskever et al., 2014). The illustration of encoder-decoder architecture is shown in Figure 2.2.



(Keselj, 2009)

Figure 2.2. Encoder-decoder model for neural response generation in dialog

The encoder transforms a sequence of embedded tokens into a sequence of hidden representations. If it is implemented by LSTM or GRU units, the temporal relation will be encoded into the hidden representations. The decoder takes previous decoder hidden state, the concatenation of the encoder hidden state at the current time and the previous decoder output, then generate a new decoder hidden state. The decoder output is computed from the decoder hidden state. What's more, the attention mechanism is a common method to improve the performance of vanilla sequence to sequence models (Bahdanau et al., 2014).

Another big problem in chat bots is that the outputs can be fluent but lack of common sense. This is because the decoder can only learn the pattern of language, but cannot really

understand the meaning of it. Some works have tried to include common knowledge into the decoder. A fully data-driven knowledge-grounded neural conversation model (Ghazvininejad et al., 2018) uses Twitter 3-turn conversations as the corpus to create the basic dialogues, uses Foursquare tips dataset (which contains restaurant comments) as the world facts. Figure 2.3 describes the knowledge-grounded model. User utterances are used to find relevant facts from the world facts set. Then the relevant facts are added to the user utterance encoder outputs and sent to the decoder for generating responses. Human judges evaluate that these outputs are significantly more informative than before.

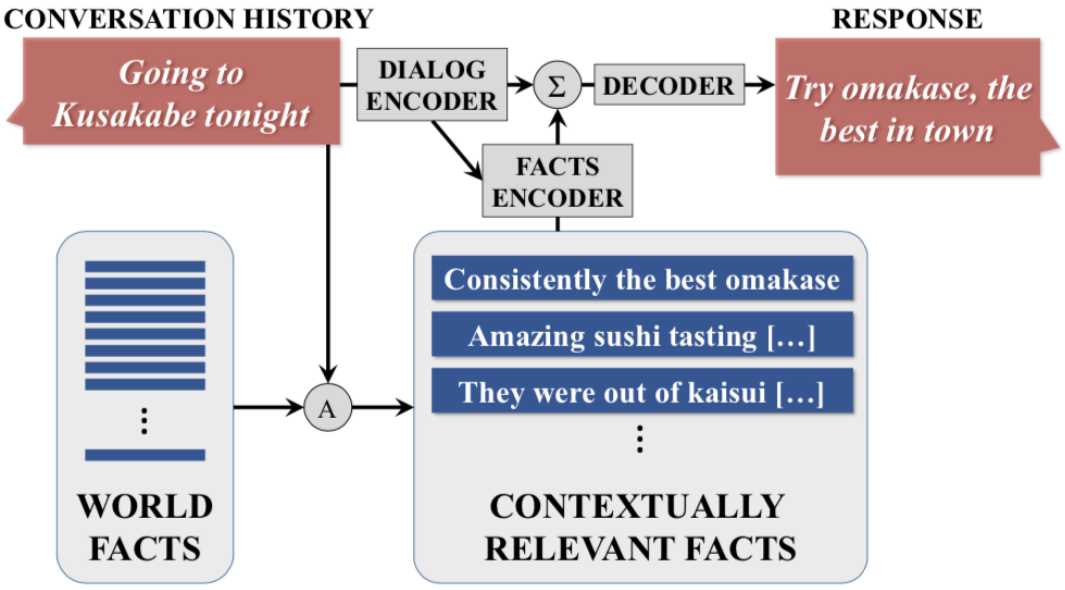


Figure 2.3. Knowledge-grounded model architecture

There is a big difference between single-turn and multi-turn chat bots. Single-turn systems only consider the previous turn to generate a response, but multi-turn systems need to consider all turns of conversion accumulated from the start. The key point to build a multi-turn system is to manage the contexts of previous turns utterances. However, if the contexts are too long or there is no need to keep all of them, we can use context segmentation (Song et al., 2016) to reduce the size of contexts.

2.3 Evaluation Metrics

The evaluation metrics are very different between task-oriented dialogue systems and chat bots.

For task-oriented dialogue systems, the major evaluation metrics are slot-value filling accuracy and task error rate. There are few needs to evaluate the generated language quality since these kinds of systems are just designed for finishing tasks. For chat bots, the mainly used method is human evaluation, since the fluency and variance in language is very hard to measure by automatic evaluation metrics. Actually, the automatic evaluation metric BLEU score (Papineni et al., 2002), which is widely used in machine translation, is reported as a very bad method to evaluate the performance of chat bots (Liu et al., 2016). Since a good evaluation metric should have a similar trend to human judgments but BLEU actually has a poor correlation with human evaluations. However, there are still some good methods to automatically evaluate the chat bot performance. One approach is to train another classifier to check if the responses come from human or a chat bot. This method is called adversarial evaluation (Li et al., 2017) and it learns the idea from the Turing Test.

CHAPTER 3

MODEL

Our dialogue system adopts pipeline-based methods, which include three major modules to extract different features and generate responses to users. In this chapter, we firstly give an overview of the task we work on. Then, we explain the framework of the proposed dialogue system. Finally, we introduce the three modules in detail.

3.1 Overview of the Dialogue System Task

The system is task-oriented and focuses on the field of restaurant reservation. The main goal of the task is to help users find a restaurant that satisfies the user's requirements and offer related information like the restaurant address and phone number to users. Figure 1.1 shows an example dialogue in DSTC2 dataset.

3.2 Framework of the Proposed Dialogue System

Our dialogue system is pipeline-based. It consists of three modules. GLAD dialogue state tracker (Zhong et al., 2018) is the first module. It is used to estimate user goals and requests from user utterances. the policy learning module is the second module and it is designed to generate system actions and restaurant information. Response generation module is the last one. We augment it with pointer-generator-network (See et al., 2017), which is an effective approach to control language generation contents.

For each turn of dialogue, the system takes the user utterance as input, and return a natural language response to the user. The process in the pipeline (shown in Figure 3.1) includes three steps:

1. Understand the user utterance, which is to extract the user goals and requests (e.g. inform(food=british), request(address)) from the user utterance. The GLAD algorithm is

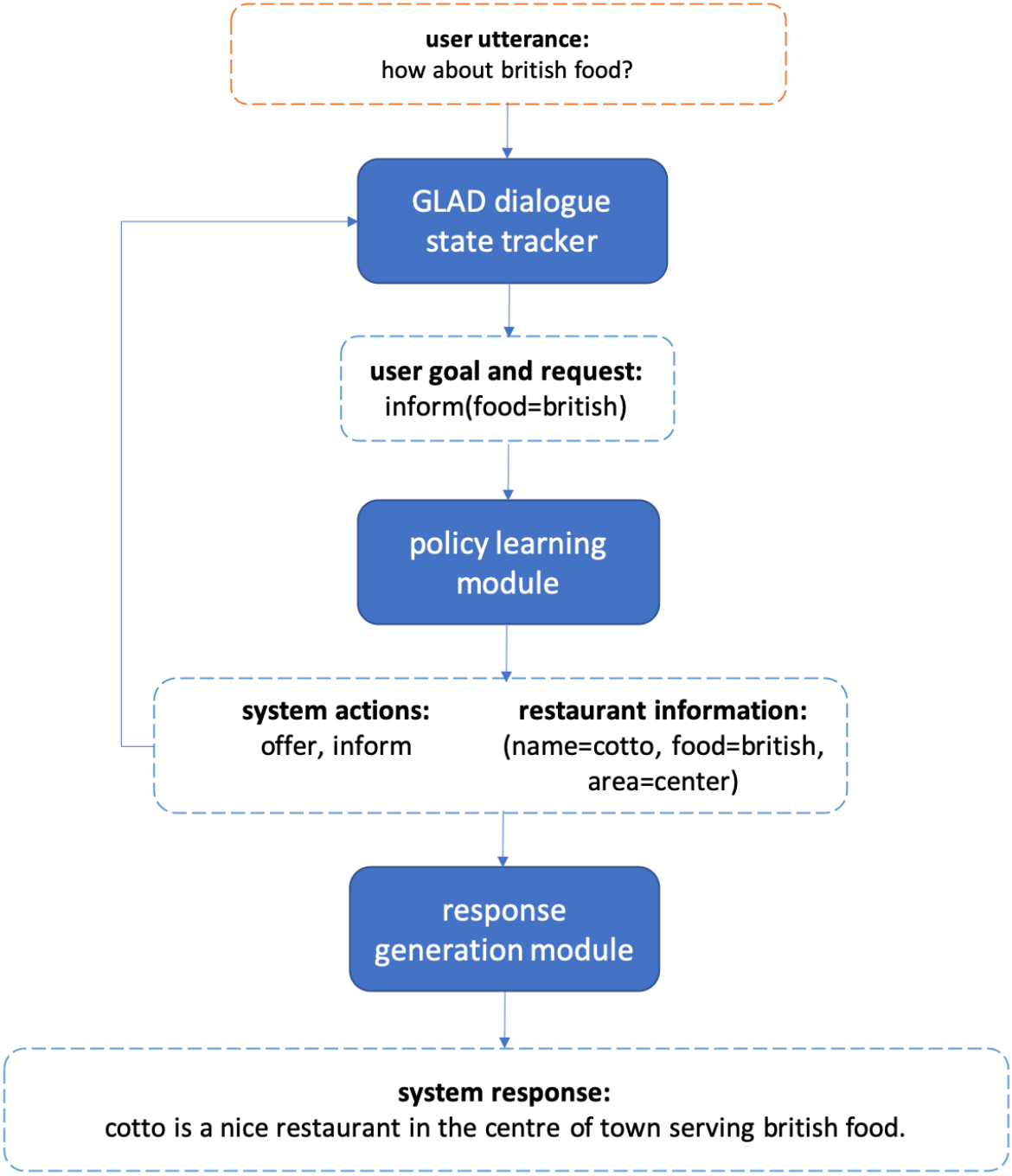


Figure 3.1. The data pipeline in the proposed dialogue system

reported performing well on DSTC2 dataset to extract dialogue states (or user goals and request) (Zhong et al., 2018). We implement the GLAD dialogue state tracker here to solve this problem.

2. Use the results obtained from step 1 to decide system actions and restaurant information. There are two uses for these actions: 1) trigger the database query to get the restaurant information according to the user goals. 2) guide the system response generation in the next module. This part of work is realized in the policy learning module.

3. According to the system actions and restaurant information, the system generates natural language responses (which will be returned to users). The response generation module is designed for this work and as mentioned above, we adopt the pointer-generator network (See et al., 2017) to improve the performance of language generation.

3.3 GLAD Dialogue State Tracker

3.3.1 An Overview of Dialogue State Tracking

Dialogue state tracking (DST) is the task to extract formatted information from natural language and manage dialogue histories. The formatted information can be in different types like food name, price range. As a convention, we usually called the formatted information as the state of dialogue and the type of formatted information is called "slot".

The state of dialogue typically includes multiple user goals and requests. We can explain it more clearly from the example dialogue described in Figure 3.1. A user goal is an object that a user would like to achieve. It starts with the term "inform" and is followed by a slot value pair (separated with an equal sign) placed in parentheses (e.g. inform(food=british)). The user request means a user would like to ask some information from the system. It starts with the term "request" and is followed by the slot (means type of information) the user wants to know (e.g. request(address)).

```
{
    "requestable": [
        "addr",
        "area",
        "food",
        "phone",
        "pricerange",
        "postcode",
        "signature",
        "name"
    ],
    "informable": {
        "food": [
            "afghan",
            "african",
            "afternoon tea",
            "asian oriental",
            ...
        ],
        "pricerange": [
            "cheap",
            "moderate",
            "expensive"
        ],
        "name": [
            "ali baba",
            "anatolia",
            "ask",
            "backstreet bistro",
            ...
        ],
        "area": [
            "centre",
            "north",
            "west",
            "south",
            "east"
        ]
    }
}
```


Figure 3.2. Part of the ontology file in DSTC2 dataset

3.3.2 Using the Domain Knowledge

For task-oriented dialogue system, it is practical to build a list that contains common words collected from the actual work scenarios. This list is called ontology and it contains the knowledge that is in a specific domain. Figure 3.2 shows part of the ontology file in DSTC2 dataset. We can use the ontology to reduce the size of possible values for each slot.

3.3.3 The Global-Locally Self-Attentive Dialogue (GLAD) State Tracker

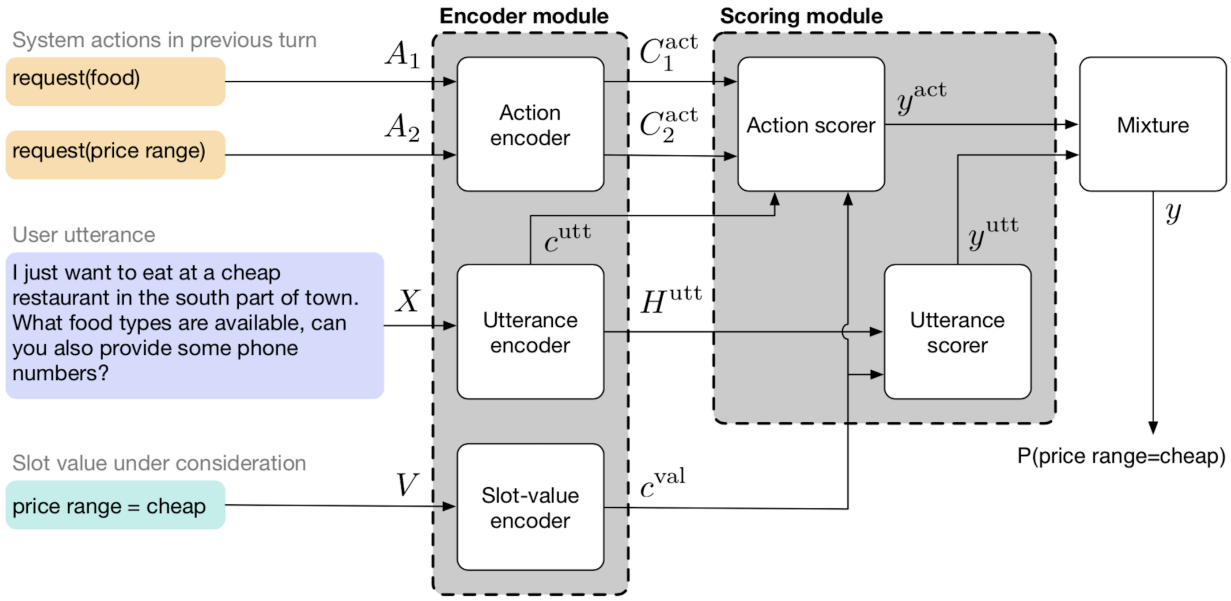


Figure 3.3. The architecture of GLAD (Zhong et al., 2018)

The DST task is a hard problem. It's really challenging to directly extract slot-value pairs from user utterances. From the ontology file, we can know all the possible values for each slot. So GLAD tracker uses every possible slot-value pair, creates one classifier for each of them. For each turn of dialogue, one classifier compares its corresponding pair with the user utterance to check whether this pair appears in the utterance or not. In detail, the classifier generates a score and if it is bigger than a threshold, then adding this slot-value pair into the dialogue state for the current turn. GLAD repeats this process for every classifier. In

this way, it converts a multi-label state prediction into a problem of binary prediction and reduces the task difficulty.

From Figure 3.3, we can see that the inputs of the GLAD tracker are system actions in the previous turn, user utterance in the current turn and one slot-value pair that is under consideration. They are encoded by separate encoders and the outputs are used to compute the score for the current slot-value pair.

3.3.4 GLAD Encoder

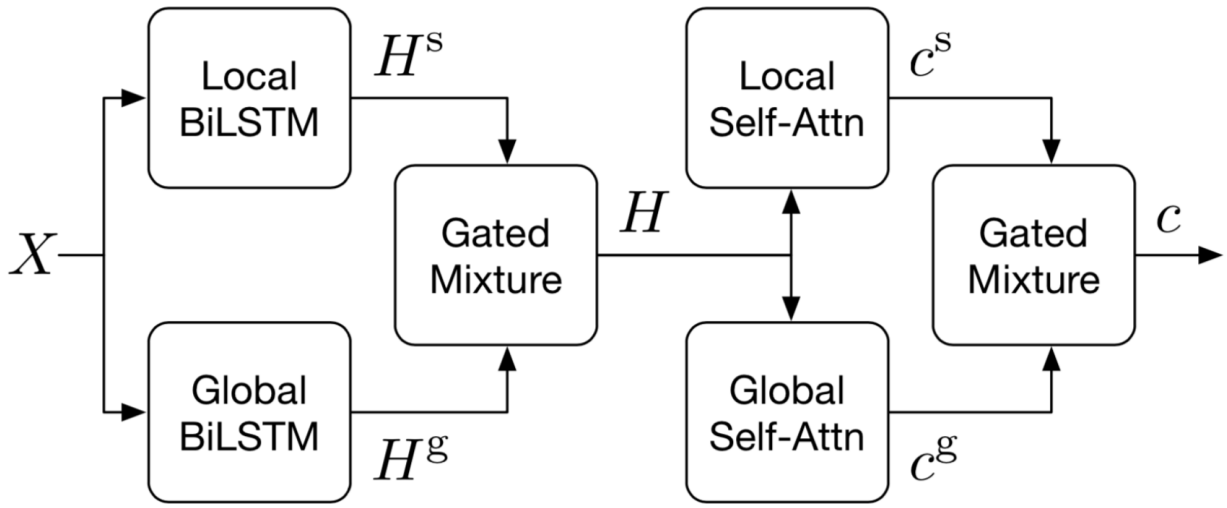


Figure 3.4. The GLAD encoder (Zhong et al., 2018)

Track the slot-value pairs that are rare to appear in the dataset is a difficult problem in DST task. GLAD tries to solve this problem by using two kinds of LSTM encoders to capture more information for each kind of slot. The structure of encoder is showed in Figure 3.4. Given the input X ($X \in \mathbb{R}^{n \times d_{emb}}$, where n is the number of words in the input, d_{emb} is the size of word embedding) and one slot, the local BiLSTM is the bi-directional LSTM that is used only for one slot. The global BiLSTM is the LSTM that is used for all slots. We need to further explain the meaning of this kind of structure. Usually, the global LSTM is

the default choice. But it cannot differentiate various slots. By using local LSTMs, GLAD can collect more information for each slot. This can help to solve the difficult problem of tracking rare slot-value pairs.

Input X is encoded by both local BiLSTM and global BiLSTM, then combines the outputs of these two LSTMs to generate a global-local encoding H, which contains the temporal relationship of X. This process can be described by the following equations:

$$H^s = \text{biLSTM}^{\text{s}}(X) \tag{3.1}$$

$$H^g = \text{biLSTM}^{\text{g}}(X) \tag{3.2}$$

$$H = \beta^s H^s + (1 - \beta^s) H^g \tag{3.3}$$

H^s, H^g, H are in the dimension of $\mathbb{R}^{n \times d_{rnn}}$, where d_{rnn} is the dimension of the LSTM, β is a learned parameter between 0 and 1 that is specific to one slot s.

Next, compute the self attention of H. Again, GLAD encoder includes both global self attention (which is used for all slots) and local self attention (which is only used for one slot). Equations 3.4, 3.5, 3.6 are used to compute both the local self attention c_s and the global self attention c_g. The diminsion of c_s and c_g are both $\mathbb{R}^{d_{rnn}}$.

$$a_i = W H_i + b \tag{3.4}$$

$$p = \text{softmax}(a) \tag{3.5}$$

$$c = \sum_i p_i H_i \tag{3.6}$$

Combine the two contexts c_s and c_g to get the final attention context c ($\in \mathbb{R}^{d_{rnn}}$). And comparing to the encoding H, context c is the summary of input X.

$$c = \beta^s c^s + (1 - \beta^s) c^g \tag{3.7}$$

As shown in Figure 3.4, we use three different encoders to encode the system actions in the previous turn, the user utterance in the current turn and one slot-value pair that is under consideration. Equations 3.1 - 3.6 describe all the computation in the GLAD encoder. Finally we get the encoding and self-attention context for the jth previous-turn system action (H_j^{act}, c_j^{act}), user utterance in the curren turn (H^{utt}, c^{utt}), and the slot-value pair that is under consideration (H^{val}, c^{val}).

3.3.5 GLAD Score Module

From Figure 3.4, we can see that there are two scoring modules. The reason for using two scoring modules is that the dialogue states can either come from the current turn user utterances or from the previous turn system actions (e.g. in the previous turn, the system asks user "would you like a restaurant in the south of town?" and the user replies "yes"). The final score will be the combination of the two scores and we use this score to decide whether to include the slot-value pair in the dialogue state or not.

Equations 3.8 - 3.11 describe computations in the utterance scorer, which is to get the attention between the user utterance H^{utt} and the slot-value pair being considered c^{val}, then use the attention context q^{utt} to score the slot-value pair.

$$a_i^{utt} = (H_i^{utt})^{\mathrm{T}} c^{val} \tag{3.8}$$

$$p^{utt} = \mathrm{softmax}(a^{utt}) \tag{3.9}$$

$$q^{utt} = \sum_i p_i^{utt} H_i^{utt} \tag{3.10}$$

$$y^{utt} = W q^{utt} + b \tag{3.11}$$

Where $p^{utt} \in \mathbb{R}^m$, $q^{utt} \in \mathbb{R}^{d_{rnn}}$, $y^{utt} \in \mathbb{R}, m$ is the number of words in the user utterance.

Equations 3.12 - 3.15 describes computations in the action scorer, which is to get the attention q^{act} between the system action self-attention context c^{act} and the utterance self-attention context c^{utt}, then use the similarity between q^{act} and c^{val} as the score.

$$a_j^{act} = (c_j^{act})^{\mathrm{T}} c^{utt} \tag{3.12}$$

$$p^{act} = \mathrm{softmax}(a^{act}) \tag{3.13}$$

$$q^{act} = \sum_j p_j^{act} c_j^{act} \tag{3.14}$$

$$y^{act} = (q^{act})^T c^{val} \tag{3.15}$$

Where $p^{act} \in \mathbb{R}^{l+1}$, $q^{act} \in \mathbb{R}^{d_{rnn}}$, $y^{act} \in \mathbb{R}, l$ is the number of system actions.

We get the final score by mixing the two scores above and normalize it through a sigmoid funciton.

$$y = \mathrm{sigmoid}(y^{act} + \omega y^{utt}) \tag{3.16}$$

Where $\omega \in \mathbb{R}$, $y \in \mathbb{R}$. As mentioned before the score y is used to compare with a threshold. If y is higher than the threshold, we will include the corresponding slot-value pair into the dialogue state. The threshold is set to a constant value and all slots have the same value (0.5) in our experiment.

3.4 Policy Learning Module

Policy network is designed to generate system actions and restaurant information. It employs the encoder-decoder architecture (Sutskever et al., 2014) and the LSTM cell is used as the recurrent unit. The inputs are the dialogue states come from the GLAD tracker. The outputs are system actions that can trigger the restaurant query and guide the response generation in the next step and restaurant information.

For a turn of dialogue, although the dialogue states that come from GLAD tracker are slot-value pairs, we take them as a sequence of words like $\mathrm{w}_1, \mathrm{w}_2, ..., \mathrm{w}_\mathrm{n}$. They are sent to the encoder to generate the encoder hidden representations $\mathrm{h}_1, \mathrm{h}_2, ..., \mathrm{h}_\mathrm{n}$.

Next, we use two decoders to separately generate the system actions and restaurant information. The system firstly predicts the system actions in the system action decoder, then predicts the restaurant information in another decoder by using the predicted system actions. The reason we use two decoders is that: the system actions are defined in the ontology file and the number of possible values for system action is small. It can be much easier to predict system actions separately comparing to predict the system actions and restaurant information together (in which case the number of candidate values for each output token can be much bigger).

The attention mechanism (Vinyals et al., 2015) is used to improve the performance of the decoder. At each timestamp t, we compute the decoder output y (which is a vocabulary probability distribution) as follow:

$$u_i = v^T tanh(W_1 h_i + W_2 \tilde{h}) \tag{3.17}$$

$$a = \text{softmax}(u) \tag{3.18}$$

$$\tilde{h}' = \sum_i a_i h_i \tag{3.19}$$

$$o = U[\tilde{h}, \tilde{h}'] \tag{3.20}$$

$$y = \text{softmax}(o) \tag{3.21}$$

Where v, W_1, W_2, U are parameters, $\tilde{h}$ is the decoder hidden state at time t. The output y is in the size of language generation vocabulary size.

After getting the predicted system actions, we can generate the restaurant information. It comes from two possible sources. The first one is from the database. If the set of predicted system actions contains the action "offer", the system will query the database according to the dialogue states and return some restaurant information. The second source is dialogue states that come from the user utterance. Since users can give some requirements of the restaurant, but the system may not be very sure about the contents of the user utterance

and need to get a confirmation from the user. For example, a user says "I'd like to find a restaurant that is near to the south church of the town." and system replies with "Are you looking for a restaurant that is in the south of town?". One important thing we need to mention is that, since there are two possible sources of the restaurant information, our system always chooses the database information as the first choice, and only if it does not exist, then use the information comes from user utterance.

3.5 Response Generation Module

Response generation (referred to as generation module for short) is the last module of the pipeline. It takes system actions and restaurant information as input, convert them into a natural language response and return it to users.

Generation module also uses the encoder-decoder architecture (Sutskever et al., 2014) and it is augmented with the attention mechanism (Vinyals et al., 2015). Again, we take LSTM cells as the recurrent units.

To generate a natural language response, the system actions determine the sentence pattern and the restaurant information should always appear in the response without any change. The attention augmented decoder can learn the match between system actions and sentence pattern, but it's hard to control the restaurant information in the response to be the same as the information comes from the policy module. We learn the idea from pointer-generator network (See et al., 2017) to solve this problem.

Our approach is to change the probability distribution for the decoder output so that we can better control the contents of the language generation. We take the restaurant information as the keyword list L, and would like to embed words in L into the system response. When decoder is predicting the output token y, for each possible word w in the generation vocabulary, we modify its probability in the vocabulary distribution p_{vocab} in the following way: if w is not in the keyword list L, we reduce its probability by multiplying

a value q_{gen} ($\in [0, 1]$); otherwise, we keep the probability of w untouched. After modifying the probabilities of all the words, we use a sigmoid function to normalize and create a new output vocabulary distribution.

$$p_{\mathrm{w}} = q_{gen} * p_{\mathrm{w}}, \mathrm{w} \notin \mathrm{L} \tag{3.22}$$

$$q_{gen} = \mathrm{sigmoid}(\mathrm{v}_1^T * o + \mathrm{v}_2^T * \tilde{h} + \mathrm{v}_3^T * c_{act}) \tag{3.23}$$

$$p_{vocab} = \mathrm{softmax}(p) \tag{3.24}$$

Where v_1, v_2, v_3 are learned parameters. o is the decoder output before using pointer-generator. $\tilde{h}$ is the decoder hidden state, c_{act} is the context vector of the system actions.

CHAPTER 4

EVALUATION

In this chapter, we firstly introduce the DSTC2 dataset and the pre-processing on data. Next, we present the training details (like hyper-parameters) in our experiments. Next, we describe our main comparison copy-network and metrics. Then the full experimental results and analyses are offered. After that, we compare between copy-network and pipeline-based models. Then we give the error analysis and discuss possible improvements to our system. Finally, we show a sample dialogue from the system outputs.

4.1 Dataset

Table 4.1. Statistics of DSTC2

Average Number of Utterances Per Dialogue	14
Vocabulary Size	1229
Training Dialogues	1618
Validation Dialogues	500
Test Dialogues	1117
Number of Entity Types (or Slot)	8
Number of Distinct Entities	452

We use the Dialogue State Tracking Challenge 2 (DSTC2) dataset (Henderson et al., 2014), which contains dialogues that are in the domain of restaurant reservation. Table 4.1 shows the basic statistics of it. All data in DSTC2 comes from the log and annotation files of a real-world dialogue system called "Cambridge restaurant system" (referred to as Cambridge system for short). Log files record all important information generated from Cambridge system. Figure 4.1 shows a sample log file. We mainly use "transcript" and "dialog-acts" to rebuild a complete dialogue. The "transcript" section is the system response. The "dialog-acts" are system actions and restaurant information got from an external database. Figure 4.2 shows part of the information in a sample annotation file. We only use "transcription"

and "goal-labels" for our project. The "transcription" section is the user utterance and the "goal-labels" are the user goals and requests. We divide the dataset into train/validation/test splits, use the train split for training model, find model weights (like weights of LSTM and linear layers) that get the best performance on validation split, and finally evaluate the model on the test split.

The DSTC2 dataset is initially designed for the dialogue state tracking task. However, in our experiments we use it to build a complete dialogue system since the log and annotation files of DSTC2 contain all the information that is needed for this object.

There are some modifications to the data since we use it in a different way comparing to its initial design. The major one is on the multi-word phrases that locate in the DSTC2 ontology file. We connect each word with underline so that the phrase is converted into a special word (e.g. we convert "fish house cuisine" into "<fish_house_cuisine>"). This operation is reasonable since these multi-word phrases are keywords, and a keyword should be naturally taken as a whole.

4.2 Training Details

The pipeline contains three modules. We firstly train the GLAD tracker alone, then freeze the parameters in this module and train the whole system together.

For the training of GLAD, we use the same hyper-parameters from the work of Zhong et al. (Zhong et al., 2018). For the training of the rest of our system, we employ cross-entropy loss, Adam optimizer (Kingma and Ba, 2014), batch size is set to 30, apply a dropout of 0.5, learning rate is equal to 0.0005. Both the embedding size and hidden size is set to 300. To avoid gradient explosion, we add a gradient clipping of 5. Different from the setting in Zhong et al.'s work (Zhong et al., 2018), we use randomly initialized embeddings instead of any pre-trained ones in both GLAD module and the rest of the system since it brings a better performance in our experiments. The number of layers for all LSTM cells is set to 2.

```
{
   …,
   "caller-id": "0a45bc863d",
   "turns": [
      …,
      {
         "output": {
            "transcript": "saint johns chop house is a nice place in the west of town and the prices are
moderate",
            "end-time": 33.75,
            "start-time": 27.4384,
            "dialog-acts": [
               {
                  "slots": [
                     [
                        "name",
                        "saint johns chop house"
                     ]
                  ],
                  "act": "offer"
               },
               {
                  "slots": [
                     [
                        "pricerange",
                        "moderate"
                     ]
                  ],
                  "act": "inform"
               },
               {
                  "slots": [
                     [
                        "area",
                        "west"
                     ]
                  ],
                  "act": "inform"
               }
            ],
            "aborted": false
         },
         "turn-index": 3,
         "input": {...
         }
      }, ...
   ],
   "system-specific": {
      "dialog-manager": 2,
      "acoustic-condition": 1
   }
}
```


Figure 4.1. Part of a sample log file for one dialogue from DSTC2 dataset. It contains all important information generated by the cambridge system when processing user utterances.

```
{
   "caller-id": "0a45bc863d",
   "turns": [
      …,
      {
         "turn-index": 1,
         "goal-labels": {
            "food": "catalan"
         },
         "transcription": "catalan food",
         "method-label": "byconstraints",
         "audio-file": "pt344x_0000938_0001054.wav",
         "requested-slots": [],
         "semantics": {
            "json": [
               {
                  "slots": [
                     [
                        "food",
                        "catalan"
                     ]
                  ],
                  "act": "inform"
               }
            ],
            "cam": "inform(food=catalan)"
         }
      },
      ...
   ],
   "task-information": {…},
   "session-id": "voip-0a45bc863d-20130325_200321"
}
```


Figure 4.2. Part of a sample annotation file for one dialogue from DSTC2 dataset. It contains user utterances transcribed by human (from a audio file) and manually-labeled user goals.

Except for the GLAD module, model weights (like LSTM weights) in the rest of the system are initialized by Glorot initialization with normal distribution (Glorot and Bengio, 2018).

4.3 Main Comparison and Metrics

We take the copy-network (Eric and Manning, 2017) as our main comparison. It also builds a dialogue system on the DSTC2 dataset, but it just uses one encoder-decoder framework (Sutskever et al., 2014) for the whole system and employs some basic attention mechanisms (Vinyals et al., 2015). Since this model is the same as our policy learning module, Equations 3.17 - 3.21 (which are the equations that describe our policy learning module) can also explain the computations in copy-network.

Copy-network takes user utterances as input and predicts system responses and API calls (API calls can trigger database query to get restaurant information that is used in response generation). Two extra mechanisms are used in copy-network. The first one is copy attention of the encoder output (Equations 3.17, 3.18) decoder. Formally Equation 3.20 is changed to $o = U[\tilde{h}, \tilde{h}', a^t_{1:m}]$, where $a^t_{1:m}$ is the concatenation of attention scores of encoder output. The second change is on the encoder inputs. According to the DSTC2 ontology file, each keyword in this file belongs to one of eight possible slots (as shown in Table 4.1). Before sending an embedded token into the encoder, copy-network checks if this token is an embedded keyword that comes from the ontology file. If the answer is true, then an extra one-hot vector (which represent the slot this token belongs) is sent to the encoder after this embedded token.

As a complete neural network based method, copy-network is much simpler than the pipeline-based architecture (our model). But pipeline approaches are more explainable and widely used in the industry to build product-level dialogue systems. Thus it's valuable to compare between the two methods on one dataset to further understand the pros and cons of each kind of model.

Our dialogue system contains three modules, we use different metrics to evaluate the performance of each module. For the GLAD module, the outputs are user goals and requests. So we use the turn-level goal accuracy and turn-level request accuracy as the metrics. These metrics can show how much formatted information is extracted from user utterances. Similarly, for the policy learning module, we use the system action accuracy and restaurant information accuracy as the metrics. For the response generation module, we employ the per-response accuracy from our main comparison copy-network. This is a very strict metric since only if each of the token in the generated response matches the corresponding token in the real one, the module output can be seen as a correct prediction.

4.4 Experimental Results

Since our system is based on a pipeline, we would like to know the effects of each module on the whole system. Table 4.2 shows the experimental results under four conditions. For the GLAD tracker, we can choose to use it or replace it with real user goals and requests. For policy learning module, we can also choose to use it or replace it with real system actions and restaurant information. We will analyze the performance of each module in the following parts.

4.4.1 Performance of the GLAD Tracker

The results of the GLAD tracker are user goals and requests, so let's check the turn-goal accuracy and turn-request accuracy first. From Table 4.2, in the column of use GLAD we can see that the turn-goal accuracy are 0.756 (use policy learning) and 0.742 (use real system actions & restaurant information) and turn-request accuracy is 0.994 (for both use or not use policy learning). Since GLAD depends on previous turn system actions (which are the output of policy learning module), it's reasonable that there is a minor difference between the two conditions.

Table 4.2. Results from our experiments under four different conditions.

	use GLAD	**use real user goals & requests**
use policy learning	turn-goal acc: 0.756 turn-request acc: 0.994 system action acc: 0.479 restaurant information acc: 0.546 system response acc: 0.371	turn-goal acc: 1.0 turn-request acc: 1.0 system action acc: 0.490 restaurant information acc: 0.549 system response acc: 0.367
use real system actions & restaurant information	turn-goal acc: 0.742 turn-request acc: 0.994 system action acc: 1.0 restaurant information acc: 1.0 system response acc: 0.675	turn-goal acc: 1.0 turn-request acc: 1.0 system action acc: 1.0 restaurant information acc: 1.0 system response acc: 0.675

Next, we need to check the effects of GLAD tracker on the whole system. In our design, the GLAD module's outputs are passed to the policy learning module to generate system actions and restaurant information. So it is reasonable to use system action accuracy and restaurant information accuracy as metrics, compare the differences between the two conditions of using GLAD tracker or not. And we also check the effects on the final system output by comparing the system response accuracies between the two conditions.

The outputs of GLAD tracker contains errors (so the turn-goal accuracy and turn-request accuracy are both less than 1.0). We would like to know if these errors have some big effects on the whole system. So we can replace the outputs of GLAD by the the real user real goals and requests and check the differences. From Table 4.2 we can see: if we use policy learning, the differences between the three mentioned metrics are very small (all differences are less than 2.5%). And if we use real system actions and restaurant information, the differences between the three mentioned metrics are even zero. Thus it's safe to reach the conclusion that: in the current stage, the GLAD tracker is not the bottleneck to the whole system.

4.4.2 Performance of the Policy Learning module

As mentioned above, we use the system action accuracy and restaurant information accuracy as the metrics to evaluate the performance of the policy learning module. From Table 4.2 we can see that: the system action accuracy of both "use GLAD" and "use real user goals and requests" are less than 0.5 and the restaurant information accuracy for both cases are little higher than 0.5. Considering that the error will accumulate along the pipeline, such poor performance in policy learning will certainly limit the performance of the whole system. This prediction is proved by the experiments. Again from Table 4.2 we can see: when using the policy learning module, the system response accuracy are 0.371 (use GLAD), 0.367 (use real user goals & requests). If we use the real system actions and restaurant information to replace the outputs of the policy learning module, the system response accuracy is significantly improved to 0.675. This shows that the policy learning module limits the performance of the whole system.

4.4.3 Performance of Response Generation

Table 4.3. Response accuracies of different models

models	response accuracy
copy-network	0.480
seq-to-seq + attn	0.023
seq-to-seq + attn + pointer	0.371
seq-to-seq + attn + pointer + real act and info	0.675

The response accuracy of copy-network (Eric and Manning, 2017) and our pipeline-based systems are presented in Table 4.3. The basic language generation model of our pipeline is a sequence to sequence model (referred to as seq-to-seq) and it is augmented with attention mechanism (referred to as +attn). +pointer means to use the pointer-generator approach

in the response generation. +real act and info refers to replace the policy learning module with real system actions and restaurant information.

There is a large gap between the performance of our model and the performance of the copy-augmented network. The main reason is the performance of policy learning module is so poor that it seriously restricts the ability of the whole system. If we replace the policy learning module with the real system actions and restaurant information, the response accuracy can reach 0.675, which is much higher than copy-network. What's more, when we only use the naive decoder to predict the response, the accuracy is only 0.023. However, if we add the pointer-generator mechanism into the decoder, the response accuracy is improved to 0.371. This proves that the pointer-generator approach is an effective way to help language generation as expected.

4.5 Comparison between Copy-network and Pipeline-based Models

As we can see from Table 4.3, copy-network has a higher response accuracy (0.480) than ours (0.371). This shows the value of neural networks: they are good at modeling a problem whose implicit patterns are hard to describe. We can say it is effective but the weakness is also clear that it's hard to explain why the performance is so good and how can we do some specific changes to the model. In the field of the task-oriented dialogue system, this is a big problem. Since we need to control the system responses as much as possible. If there is an error, we need to know the reason for it. But for a neural network, the whole system is a black box and it's too hard to make a specific change. Such weakness decides that the complete neural network models are far from practical applications in the area of task-oriented dialogue systems.

And for our pipeline-based models, although the performance is not very good right now, we can clearly know which part of it can be the bottleneck and make some improvements in the future. If we can select a good model, the policy learning will not be the bottleneck to

the whole system. What's more, in our method, a big task is reduced into several smaller and easier ones comparing to the task met by our opponent. And it's reasonable that the performance of our system should be at least as good as the complete neural network-based methods. On the other hand, the core feature of pipeline-based methods is modularity. This guarantees that we can get very strong control over the whole system, which is the major requirement in task-oriented dialogue systems. As a result, we can say that the pipeline-based methods should be more suitable to be used in task-oriented dialogue systems. The performance of these two kinds of methods can be close. However, on the topic of taking better control of the whole system, pipeline-based methods are definitely stronger than complete neural network based approaches.

4.6 Error Analysis and Further Improvements

There are three possible sources of errors: dialogue states generated from the GLAD tracker, system actions and restaurant information predicted from the policy learning module, system responses generated from the language generation module. As shown in the previous performance evaluations, the main limitations of our system locate in the policy learning section. If we can improve the performance of policy learning, the whole system can get much better results. Thus we focus on analyzing the errors that come from policy learning module and try to give a solution to this part.

The major error in the output of policy learning module is that: there are too many wrong predictions in the system actions. And we find two main reasons for this problem. Firstly, the data is noisy. Since it comes from an old real-world system, the logic to determine system actions is not crystal clear. Secondly, the model we use in our policy learning section is not suitable for this problem. In detail, the inputs of policy learning are dialogue states which are slot-value pairs. In our design, we take one dialogue state as a sequence like a sentence, send it to an LSTM encoder-decoder neural network (Sutskever et al., 2014),

and finally generate system actions as a sequence. LSTM is good at capturing temporal relation contained in the sequences. However, both of our input slot-value pairs and the output system actions don't show strong temporal relation, which means the order is not important. Thus LSTM encoder-decoder neural network cannot show its advantages in this scenario and we should try some other approaches.

To increase the prediction accuracy in system actions, we can employ a simpler model and include reinforcement learning into the training process. The model is a one-hidden-layer neural network with tanh activation function and uses a softmax function as the output layer to predict system actions. The reason we choose a model that is simpler than before is that we don't need to capture temporal relation in the inputs and outputs. And since the size of possible system actions is not very big, a one-hidden-layer neural network can deal with this problem. What's more, the most important improvement is that: we use a two-phase training strategy to train this neural network. That means firstly we train it through supervised learning like before. Then we include reinforcement learning to get a better generalization. This has been reported to be an effective method to find a policy on a noisy dataset (Su et al., 2016).

4.7 Sample Dialogue Outputs Analysis

Figure 4.3 shows a sample dialogue output of our dialogue system (don't use any annotations to replace modules). We can see that except the first turn, the rest of the responses are all predicted correctly. The reason is our system uses the pointer-generator network to improve the ability to embed keywords. The result proves the designed algorithm is effective. And for the first turn, since the system actions are used to guide the response generation, the wrong predicted system action leads to a wrong response.

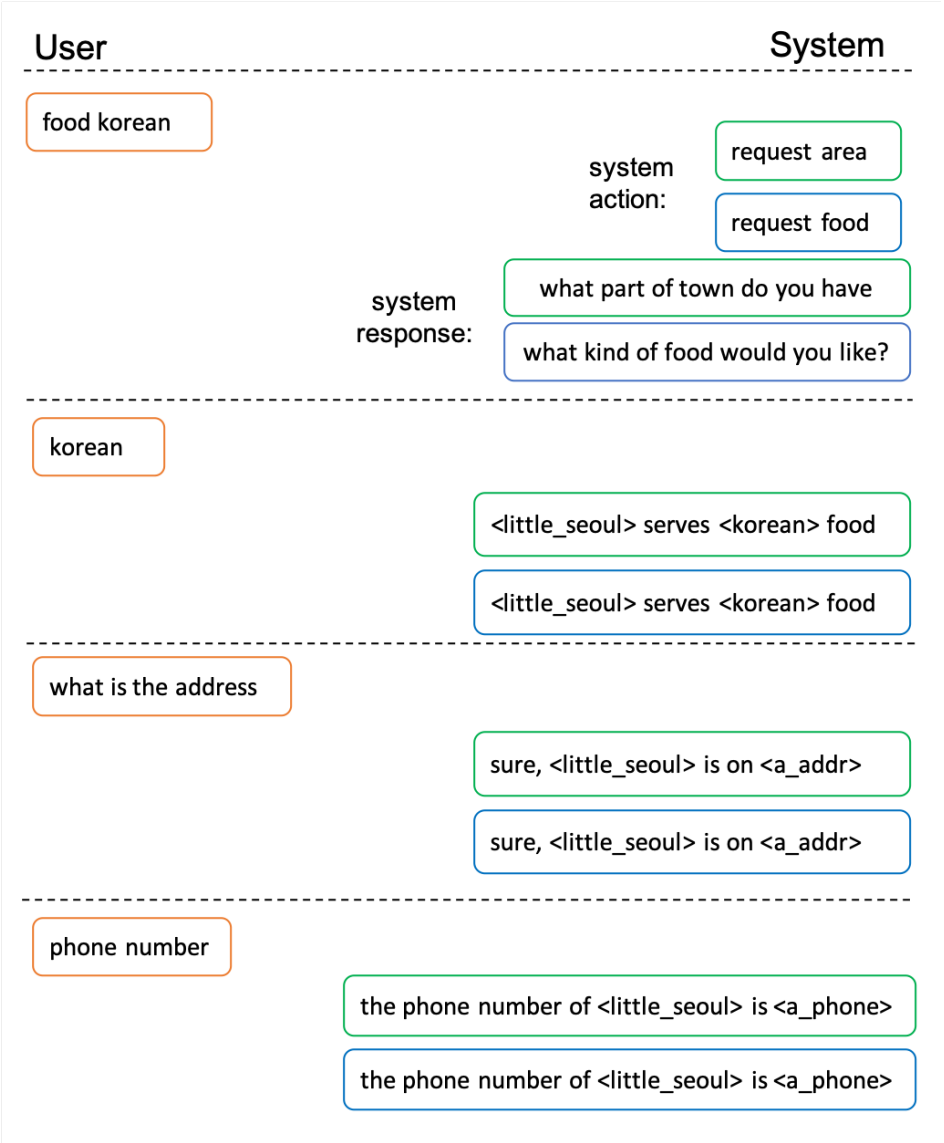


Figure 4.3. A sample output dialogue. The predicted system outputs are in green line rectangles. The real system outputs are in blue line rectangles.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

We build a pipeline-based task-oriented dialogue system on the DSTC2 restaurant reservation dataset. Our model contains three modules: a GLAD dialogue state tracker, a policy learning module, a natural language response generation module. Each of them is a smaller and easier task comparing to the original dialogue system task. Thus we can be sure that it's highly probable to find some good models to improve their performances and finally make our system to be at least as good as the copy-network (which is our main comparison and it employs an encoder-decoder neural network so it's a complete neural network based model). On the other hand, the characteristic of task-oriented dialogue systems requires to have strong control over the whole system. Copy-network cannot achieve this goal since it takes the whole system as a black box. However thanks to the modularity, pipeline-based methods are naturally better on this point. Thus it's easy to get the conclusion that we should use pipeline methods to build a task-oriented dialogue system, rather than complete neural network based approaches like copy-network.

Another contribution of our work is that we introduce a method to better control the contents of language generation. By using the restaurant information as pointers, we can increase the probability of keywords in the output vocabulary distribution. This mechanism significantly increases the system response accuracy from 0.023 to 0.371.

Future work should focus on increasing the performance of policy learning module. We have seen that by using the real system actions and restaurant information, the system response accuracy can reach almost 0.7. Comparing to the result we have now (just 0.371), there is a huge room for improvement. We can try a simple neural network model and include reinforcement learning into the training process since reinforcement learning will help to get a much better generalization on noisy data.

REFERENCES

Bahdanau, D., K. Cho, and Y. Bengio (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Cheyer, A. and D. Guzzoni (2014, March 18). Method and apparatus for building an intelligent automated assistant. US Patent 8,677,377.

Chung, J., C. Gulcehre, K. Cho, and Y. Bengio (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Clark, A. (2003). Pre-processing very noisy text. In *Proc. of Workshop on Shallow Processing of Large Corpora*, pp. 12–22.

Eric, M. and C. D. Manning (2017). A copy-augmented sequence-to-sequence architecture gives good performance on task-oriented dialogue. *arXiv preprint arXiv:1701.04024*.

Ghazvininejad, M., C. Brockett, M.-W. Chang, B. Dolan, J. Gao, W.-t. Yih, and M. Galley (2018). A knowledge-grounded neural conversation model. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Glorot, X. and Y. Bengio (2018). Understanding the difficulty of training deep feedforward neural networks. 2010. *Received February 12*.

Goddeau, D., H. Meng, J. Polifroni, S. Seneff, and S. Busayapongchai (1996). A form-based dialogue manager for spoken language applications. In *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP'96*, Volume 2, pp. 701–704. IEEE.

Hashemi, H. B., A. Asiaee, and R. Kraft (2016). Query intent detection using convolutional neural networks. In *International Conference on Web Search and Data Mining, Workshop on Query Understanding*.

Henderson, M., B. Thomson, and J. D. Williams (2014). The second dialog state tracking challenge. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pp. 263–272.

Hochreiter, S. and J. Schmidhuber (1997). Long short-term memory. *Neural computation 9*(8), 1735–1780.

Keselj, V. (2009). Speech and language processing daniel jurafsky and james h. martin (stanford university and university of colorado at boulder) pearson prentice hall, 2009, xxxi+ 988 pp; hardbound, isbn 978-0-13-187321-6, $115.00.

Kingma, D. P. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Konstan, J. A., B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl (1997). Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM 40*(3), 77–87.

Li, J., M. Galley, C. Brockett, G. P. Spithourakis, J. Gao, and B. Dolan (2016). A persona-based neural conversation model. *arXiv preprint arXiv:1603.06155*.

Li, J., W. Monroe, T. Shi, S. Jean, A. Ritter, and D. Jurafsky (2017). Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547*.

Liu, C.-W., R. Lowe, I. V. Serban, M. Noseworthy, L. Charlin, and J. Pineau (2016). How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. *arXiv preprint arXiv:1603.08023*.

Lowe, R., N. Pow, I. Serban, and J. Pineau (2015). The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. *arXiv preprint arXiv:1506.08909*.

Lu, Z. and H. Li (2013). A deep architecture for matching short texts. In *Advances in neural information processing systems*, pp. 1367–1375.

Mesnil, G., X. He, L. Deng, and Y. Bengio (2013). Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *Interspeech*, pp. 3771–3775.

Mrkšić, N., D. O. Séaghdha, T.-H. Wen, B. Thomson, and S. Young (2016). Neural belief tracker: Data-driven dialogue state tracking. *arXiv preprint arXiv:1606.03777*.

Papineni, K., S. Roukos, T. Ward, and W.-J. Zhu (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 311–318. Association for Computational Linguistics.

Ritter, A., C. Cherry, and B. Dolan (2010). Unsupervised modeling of twitter conversations. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 172–180. Association for Computational Linguistics.

Ritter, A., C. Cherry, and W. B. Dolan (2011). Data-driven response generation in social media. In *Proceedings of the conference on empirical methods in natural language processing*, pp. 583–593. Association for Computational Linguistics.

See, A., P. J. Liu, and C. D. Manning (2017). Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*.

Serban, I. V., R. Lowe, P. Henderson, L. Charlin, and J. Pineau (2015). A survey of available corpora for building data-driven dialogue systems. *arXiv preprint arXiv:1512.05742*.

Song, Y., L. Mou, R. Yan, L. Yi, Z. Zhu, X. Hu, and M. Zhang (2016). Dialogue session segmentation by embedding-enhanced texttiling. *arXiv preprint arXiv:1610.03955*.

Su, P.-H., M. Gasic, N. Mrksic, L. Rojas-Barahona, S. Ultes, D. Vandyke, T.-H. Wen, and S. Young (2016). Continuously learning neural dialogue management. *arXiv preprint arXiv:1606.02689*.

Sutskever, I., O. Vinyals, and Q. V. Le (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112.

Vinyals, O., Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton (2015). Grammar as a foreign language. In *Advances in neural information processing systems*, pp. 2773–2781.

Wang, H., Z. Lu, H. Li, and E. Chen (2013). A dataset for research on short-text conversations. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 935–945.

Wen, T.-H., M. Gasic, N. Mrksic, P.-H. Su, D. Vandyke, and S. Young (2015). Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*.

Wen, T.-H., D. Vandyke, N. Mrksic, M. Gasic, L. M. Rojas-Barahona, P.-H. Su, S. Ultes, and S. Young (2016). A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*.

Williams, J. D. (2014). Web-style ranking and slu combination for dialog state tracking. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pp. 282–291.

Yan, Z., N. Duan, P. Chen, M. Zhou, J. Zhou, and Z. Li (2017). Building task-oriented dialogue systems for online shopping. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Zhong, V., C. Xiong, and R. Socher (2018). Global-locally self-attentive dialogue state tracker. *arXiv preprint arXiv:1805.09655*.

Zhou, L., J. Gao, D. Li, and H.-Y. Shum (2018). The design and implementation of xiaoice, an empathetic social chatbot. *arXiv preprint arXiv:1812.08989*.

BIOGRAPHICAL SKETCH

Yize Pang got his master's degree in Computer Science at The University of Texas at Dallas. He received his bachelor's degree at the University of Science and Technology of China.

CURRICULUM VITAE

Yize Pang

EDUCATION

University of Texas at Dallas, Richardson, TX
Master of Science in Computer Science — *Aug. 2017-May. 2019*

University of Science and Technology of China, Anhui, China
Bachelor of Engineering in Computer Science — *Sept. 2012 -Jun. 2017*

PROFESSIONAL EXPERIENCE

Human Language Technology Research Institute, UT Dallas
Lab Assistant — *Oct. 2018-Apr. 2019*

- Built a pipeline-based dialogue system in the field of restaurant reservation (Python, PyTorch).
- Implemented GLAD algorithm to extract key information from user transcripts.
- Designed the dialogue management module to generate dialogue acts.
- Created the system response decoder through using LSTM and pointer-generator network.

Baidu, Beijing, China
Software Engineer Intern — *Jun. 2018-Aug. 2018*

- Developed Zhangwo App V2.0, an enterprise income analysis program, for China Union (Java).
- Implemented multi-turn dialogue by using Baidu UNIT, Baidu speech platform.
- Built a WeChat Mini Program and a J2EE web system with Spring Boot as the backend service.

PROJECTS

High concurrency online shopping website (Java) — *Aug. 2018-Oct. 2018*

- Created the website with Spring Boot, Mybatis, Redis.
- Implemented the MD5 password encryption and distributed session.
- Used JMeter to simulate high concurrency scenarios for stress testing.
- Improved the system through page, URL and object caching.

Information rights management system (Java) — *Jan. 2018–Apr. 2018*

- Used Spring Boot, Redis to construct the backend of the system.
- Designed the database according to the enterprise organizational structure.
- Developed the authentication and access-control by using Spring Security.

Spam filtering (Python) — *Jan. 2018–Apr. 2018*

- Implemented Naïve Bayes classifier to make e-mail filtering.
- Used bag of words feature to compute the probability that an email is or is not spam.

SKILLS

- Programming Languages: Python, Java, JQuery, Node.js, PHP, Spark, SQL
- Software: Spring Boot, Redis, MySQL, MongoDB, Hadoop, Spark, PyTorch, Google DialogFlow

AWARDS & ACTIVITIES

- Volunteer in the "2017 Texas BEST and UIL State Robotics Championship" — *Nov. 2017*
- Champion of the USTC champion league football match — *Apr. 2013*