ADL Final Project Report

組別：2 組員：b07902013 陳建豪、b07902022 張鈞堯、b07902054 林子權

Abstract

DST

In our first task - dialogue state tracking, we aim to find out the correct state (slot-value) for each turn in each dialogue. We use a pre-trained T5 model to generate the corresponding state. We also try different kind of setting to imporve our performance. In the end, this method reaches a good performance on seen domain while there's still an improvement on unseen domain.

NLG

In our second task - natural language generation(NLG), we aim to add engaging chit-chat to the existing system responses, hoping this will make task oriented responses more vivid and attractive. We train two independent classifier to score the additional chit-chat at beginning and end respectively, and fine-tuned a GPT-2 model to generate chit-chat. This method is good enough to generate interesting chit-chat at the begining or the end of the responses, and also won't defect readibility at the same time.

Introduction

DST

Dialogue state tracking has been a popular task in these years. There are also many different methods to handle this task. In this project, we've tried many kinds of public models. However, some methods require complicated preprocessing while the others required more information for testing. Finally, we've picked **T5DST** as our final method. The detail of T5DST methods will be describe in the Approach section.

NLG

The concept of adding additional chit-chat to task oriented system responses is presented in *Adding Chit-Chat to Enhance Task-Oriented Dialogues*. In this paper, they generate additional chit-chat by Blenderbot and GPT-2. After filting out some bad candidates by RoBERTa binary classifier, they applied human evaluation to label the remaining chit-chat candidates to be good or bad, which is time-consuming and has a huge human workload. Instead of evaluating by any kind of manpower, we only train two classifiers to evaluate how good the additional chit-chat is. The classifiers should directly choose out the final additional chit-chat. One of the classifier is responsible for evaluating chit-chats adding at the beginning, the other is for evaluating the chit-chats adding at the end. With the two classifiers, we are able to give each chit-chat candidate a certain evaluation score, which means that we could simply generate a huge amount of candidates and take the one with the highest score to be our answer. This is not only more easily to implement but also perform very well in practice.

Approach

DST

The concept of T5DST is presented in [Leveraging Slot Descriptions for Zero-Shot Cross-Domain Dialogue State Tracking]. Here, we only explain how we adapt to T5DST method.

Preprocessing

First, the preprocessing is similar to other DST model. For each dialogue, we keep the id, domains in the original data. For each turn with user speaker in the dialogue, we construct a new turn with the following items.

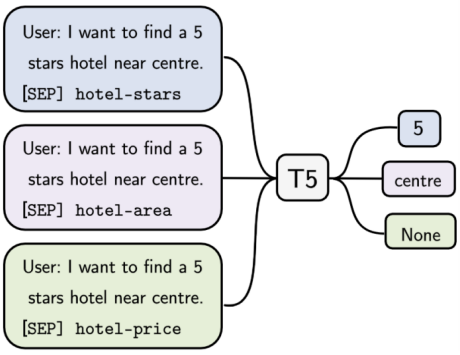
1. The system utterance of the previous turn with system speaker. If the current turn is the start turn, the system utterance will be "none".
2. The user utterance of the current turn with user speaker.
3. The state of the current turn with user speaker. Note that we don't use the active intent as the training feature. So the active intent here is always "none".

```
{
  "system_utterance": "what genre do you want?",
  "user_utterance": "i would like to see a detective movie.",
  "state": {
    "active_intent": "none",
    "slot_values": {
      "media_2-genre": "detective"
    }
  }
}
```

Second, we extract the ontology from the data. For each domain-slot type, we find all the slot values appear in the data. That is, the ontology file contains all the possible slot values for each domain-slot type.

Finally, we extract the slot description from `schema.json`. Also, if the domain-slot type is categorical, we extract there possible values, too.

Training



The pre-trained model we use the T5-small model. As shown in figure, the T5 encoder takes dialogue history and slot name as input, and T5 decoder generates the corresponding slot value. That is, at turn t, the encoder will take

$$C_t [SEP] s_i$$

as the input. C_t is the dialogue history at turn t.

$$C_t = \{U_1, S_1, \ldots, U_{t-1}, S_{t-1}, U_t\}$$

U_t, S_t is the user/system utterance at turn t, respectively. s_i is the i_{th} domain-slot type in this turn. Then, the decoder will generates the corresponding slot value v_i. The loss we want to minimize is the negative log-likelihood of v_i given C_t and s_i. That is,

$$L = -\sum_{i}^{n} \log p(v_i | C_t, s_i)$$

where n is the number of slots to be tracked.

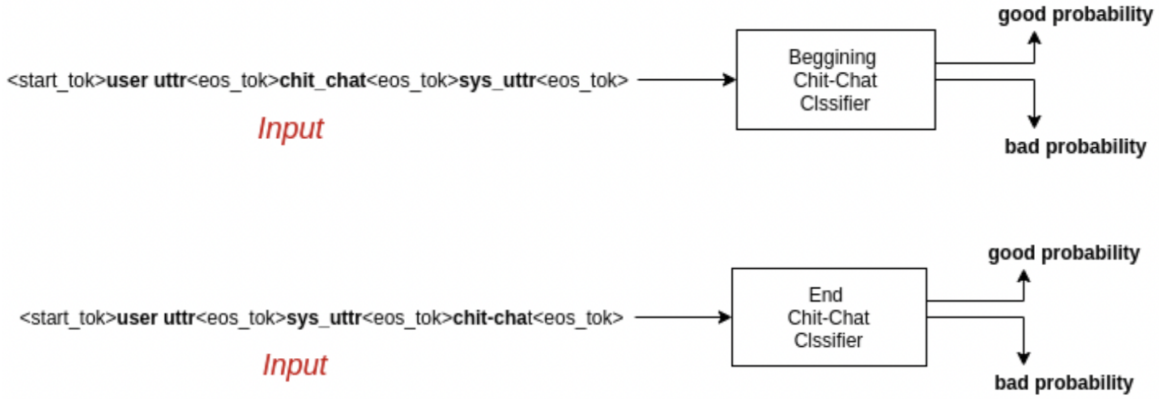
Testing

The processing of testing is similar to training. After the generation is finished, we go through all the generated slot value. If the value is "none", it means that the domain-slot type doesn't appear at this turn in this dialogue. We ignore this value. Finally, we choose the last-appear value as our slot value prediction.

NLG

First, we have to train two classifiers, one for evaluating the chit-chat adding at the beginning of the system response, one for evaluating the chit-chat adding at the end of the system response.

The way we train these two classifiers is very intuitive. Because each additional chit-chats in dataset have been labeled as good or bad, so we can train the classifiers by following the model diagram below:



In simple words, we concatenate user utterance, chit-chat, system utterance to be input data. Then, we feed the input data into a pretrained BERT model and output a vector with size 2, which represents the probability of the chit-chat to be good and bad respectively. Finally, minimizing cross entropy loss between model's output and ground truth to fine-tune the BERT classifier.

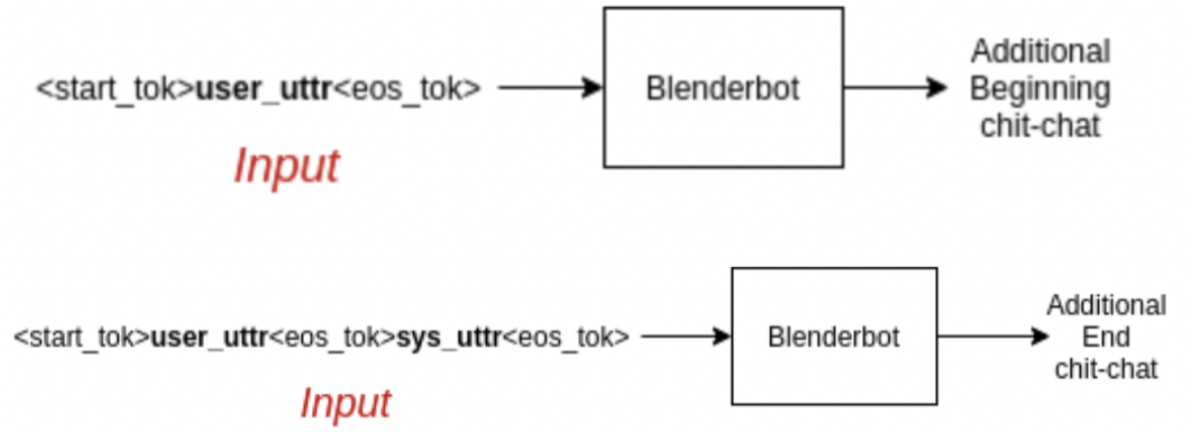
Once we have the chit-chat classifiers, the only thing we should do is to generate some chit-chat candidates and get the evaluation score by feeding them into the corresponding classifier depending on the chit-chat position.

We have tried two method to generate chit-chat candidates.

Method 1: Directly use a pretrained Blenderbot

In this method, we simply use a pretrained Blenderbot without any fine-tuning to generate chit-chat candidates.

The following diagrams show how we generate additional beginning chit-chat and additional end chit-chat respectively:

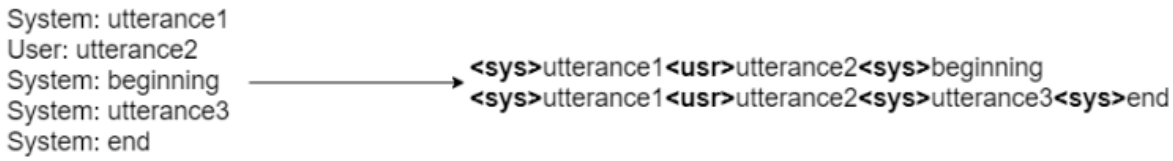


We use beam search to generate some chit-chat candidates, and first filter out those with "You're welcome" in it. Then, we could pick the one with the highest good probability to be our final answer.

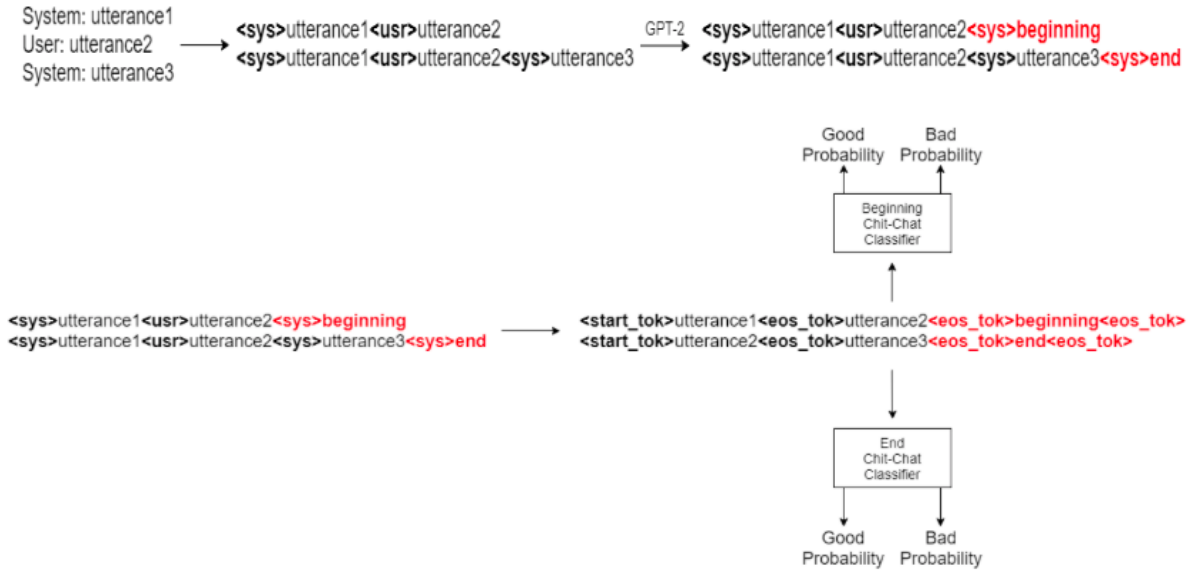
If the good probability of candidates are all below the threshold, we choose not to add any additional chit-chat to the system response.

Method 2: Fine-tuning a GPT-2 model

In this method, we fine-tune a pretrained GPT-2 model on transformers to generate chit-chat candidates. First, we introduce 2 new tokens: **<sys>** and **<usr>**, which indicates the speaker of the following utterance is system or user, respectively. Then, we use the dialogue history and the added tokens to generate training and testing data. Each training data consists of utterances until a chit-chat, with each utterance prepended with <sys> or <usr>(depending on the speaker). After all dialogues in train are processed, we use all processed data as input and label to fine-tune our GPT-2 model. The following figure shows an example of training data.



Testing data are obtained in a similar way to training data, except that there are no chit-chats in testing data. Each testing data consists of at most 10 utterances in a dialogue, with each utterance prepended with <sys> or <usr>(depending on the speaker). Then, we use these data as input for fine-tuned GPT-2 model to generate either start or end chit-chats. It's notable that if the speaker of last utterance in a data is user, the generated sequence is viewed as start chit-chat, while the generated sequence is viewed as end chit-chat if the speaker of last utterance in a data is system. We generate 5 sequences for each data, and these generated sequences will be forwared to classifiers above to calculate the good probability. Among all generated sequences, we first filtered out sequences whose good probability is lower than 0.5 and picked the sequence with highest good probability. The following 2 figures show our pipeline for generation and evaluation of sequences.



Experiments

DST

Hyperparameters

Details can be found in `config.py`. Here, we only list the parameters needed for explanation.

1. optimization algorithm: Adafactor
2. learning rate: 1e-4
3. batch size: 128 (original 8 but with gradient accumulation step 16, 8 * 16 = 128)
4. epoch: 5

Training

We've used two different training strategy.

1. Full-shot training: We use all of the data to train our model, and we choose our model based on the lowest validation loss.
2. Seperated training: Since the data can be separated into dstc8-like and MultiWoz-like, we split the data into two parts and we first train on the dstc8-like data (2 epoch), then we train on the MultiWoz-like data (2 epoch). Finally we train on all of the data (1 epoch). By doing so, we can set larger original batch size and smaller gradient accumulation step. Thus, it can speed up the training process.

Result

The result is based on the kaggle public scoreboard. (seen/unseen)

1. Full-shot training: (0.28675/0.05243)
2. Separated training: (0.26903/0.05243)

From the results, we can see that though separated training reduce the training time, the performance is not as well as full-shot training. We speculate that the model in separated training still needs more epoches on full-data training to learn the relationship between domain-slot type.

Also, the performance on unseen domain is not so good as it on seen domain. We speculate that the solution is on the way we preprocess the ontology and slot description. We've tried different kinds of format of slot description. But the performance on unseen domain reamins the same. Perhaps we can see other's method during the QA sesstion.

NLG

Chit-Chat Classifier

We have tried two different model structure to build our chit-chat classifier.

1. Separate chit-chat classifier to be two model, one for chit-chat added at the beginning, one for chit-chat added at the end. (totally same as we have descripted above)
2. Use only one classifier to classify not only chit-chat added at the beginning but also chit-chat added at the end.

The experiment result shows that the first method outperforms the second one by approximate 2% of overall classification accuracy on validation set, and the validation loss also converges at a lower value.

Number of Chit-Chat Candidates generated

In simple words, we generate chit-chat by beam search and pick the top N highest probability result to be our candidates. The parameter N can slightly affect our final answer: when N is too small, it is very likely that the candidates are all general answers, like "You're welcome" or "That sounds great!", which is not we would like to see.

Fine-tuning GPT-2

- config for training
    - Epochs: 3
    - Batch size: 1
        - Gradient accumulation step: 128
    - Optimizer: AdamW from huggingface/transformers
        - Learning rate: 1e-5
        - Linear warmup steps: 100
    - Gradient clipping value: 2.0
    - Weight decay: 1e-3, except for layer normalization and bias
    - Policy gradient
        - Reward = 5 for those chit-chats labeled as good
        - Reward = 1 for those chit-chats labeled as bad
    - full dialogue history
    - fp16 for acceleration
- config for generation
    - beam search with num_beams = 5
    - sampling rather than greedy
    - generate 5 sequences for each data
    - at most 10 recent utterances
- comparison to Blenderbot

	Blendorbot	Fine-tuned GPT-2
Generated sequence	Ambiguous, conservative	Specific, possibly inappropriate
Resource	Less time and memory	More time and memory

Conclusion

DST

In this final project, we have successfully applied T5DST method to the DST task. And the performance on seen domain is good. However, the performance on unseen domain is not so good as it on seen domain. We're looking forward to the QA session so that we can see other team's method and how they handle the unseen domain. If possible, we can apply their method to see if there's an improvement.

NLG

In this final project, we successfully modified the method which was mentioned in *Adding Chit-Chat to Enhance Task-Oriented Dialogues* by training two classifiers to give each additional chit-chat candidates a score, and got a pretty good result. This modified method not only can effectively make the dialogue more vivid, but also won't defect the readibility and humanlike of the system responses. Most importantly, this method doesn't require any human evaluation, which can significantly reduce the human workload. So, it might be a good way to generate additional engaging chit-chat.

Work Distribution

b07902013 陳建豪：NLG GPT-2 Method b07902022 張鈞堯：DST b07902054 林子權：NLG BlenderBot Method