CS388 Project 2: Semantic Parsing with Encoder-Decoder Models

Yixuan Ni, yn2782

Abstract

In this project, I implement the encoder-decoder model for solving the semantic parsing task. I implement three types of attention mechanism to improve the performance. I use scheduled sampling and batch training to further improve the model and facilitate the training process. I experiment with different parameter settings, and the best model has achieved 67.7% denotation match rate, 85.2% token-level accuracy and 60.0% exact match rate on the development set.

1 Introduction

The goal of this project is to implement an encoder-decoder model for semantic parsing. Semantic parsing is the task of converting a natural language expression to its logical forms such as lambda calculus or lambda-DCS, which are not ambiguous and can be easily processed by machines.

In this project, the problem is tackled with sequence-to-sequence models. Such model contains an encoder neural network that forms a representation of the input natural sentence, and a decoder neural network that generates the output based on the representation formed by the encoder. Attention mechanism can be used to enhance this model. The basic encoder-decoder architecture has the problem that the information at earlier time-steps is lost over time when processing long sequences. Attention mechanism provides a way for the decoder to focus more on certain parts of the input sequence so that more relevant information is retrieved when predicting the output.

The dataset I use is the Geoquery dataset (Wong and Mooney, 2006), in which natural language queries are paired with Prolog formulas that executed on a knowledge base to find the answers of queries.

The project includes three main parts: implementing the basic sequence-to-sequence algorithm, implementing the attention mechanism, and extensions to improve the models. In the next sections, I will explain in details the implementations, the experiments I have done and compare the results.

2 Implementation

For the basic sequence-to-sequence algorithm, the encoder LSTM network has been provided. For the decoder, I first use an embedding layer to embed the input. Then the input and the hidden state are feed into an RNN layer, which generates an output and a new hidden state to serve as the input at the next time-step. A softmax layer is then added on top of the RNN layer to produce the output logits and compare with the target output to give the loss.

During training, the sentence and target pairs are feed into the model one by one. The encoder will output a hidden state from the embedded sentence, which is called a context vector. To start decoding, a tensor of the the start-of-sentence (SOS) symbol is used as the decoder input and the context vector is used as the initial decoder hidden state. Then the decoder will start to produce the output sequentially. It will stop when it has generated the end-of-sentence (EOS) symbol, or when the length of the output reaches a maximum limit (65 words). Then the loss is calculated and back-propagated through the whole networks.

2.1 Attention Mechanism

For the attention mechanism, I follow the definitions in (Luong et al., 2015) and refer to the implementation in the tutorial (Spro). I only use the global attention mechanism, where all the hidden states of the encoder are used to derive the context vector. First an alignment vector is calculated by taking the softmax of scores of current target hidden state and each source hidden state. There are three types of scores: dot, general and concat. Dot score is the dot product of the target state and the source state; general applies a linear transformation on top of the dot score; concat first applies a linear transformation on the concatenation of the two states, then calculates the dot product of the concatenation with a weight vector. The alignment vector is then used as the weights to calculate a weighted average of the context vector. Finally, the context vector and the context vector are feed through a concatenation layer and then a softmax layer to produce the output.

2.2 Extension

I implement two extensions to improve the model performance and facilitate the training time.

Teacher Forcing and Scheduled Sampling The approach of taking the output of the decoder as the input at the next step has a problem that

Model	Exact Match	Token-level Acc	Denotation Match
Basic	0.058	0.590	0.075
Attention (general)	**0.600**	**0.852**	**0.617**
Attention (dot)	0.542	0.846	0.575
Attention (concat)	0.383	0.799	0.425
Teacher forcing rate = 0	0.500	0.848	0.542
Teacher forcing rate = 0.5	**0.600**	**0.852**	**0.617**
Teacher forcing rate = 1	0.425	0.820	0.508
Batch size = 2	**0.425**	**0.820**	**0.508**
Batch size = 10	0.367	0.778	0.400

Table 1: Performance on the development set of different models.

errors can accumulate along the sequence generation. One mistake can lead the generation to a completely wrong direction. A way to solve this is to use teacher forcing, where the known target tokens are used as input and guide the whole generation process. Although the model performs better during training, when translating unknown input sequence, there are no target values, and the model's performance reduces due to the lack of guidance. A better approach is to use scheduled sampling, where there is a probability that teacher forcing is used. In my implementation, I can set a probability so that for each training batch, the model will choose whether teacher forcing is used. Probability of 0 corresponds to the original approach, and probability of 1 corresponds to the pure teacher forcing approach. I will show in the results section that the use of scheduled sampling can improve the performance significantly.

Batch Training I implement batch training to facilitate the training process. For the decoder, the input is shaped as $[num\_time\_steps, batch\_size, decoder\_input\_size]$ and the decoder will generate a batch-sized vector as the output at each timestep. I use a counter to count the number of EOS tokens that have been generated, and stop the generation when the counter reaches the batch size, which is a sign that all of the output sequences in the batch have finished. Although it seems to be a problem dealing with the padding tokens when one sequence has finished but others are not, empirically, this approach does not harm the performance. The reason is padding tokens are always generated after the EOS token, and the model, if learned correcly, will aovid generating padding tokens before EOS token, hence the sequences are generated normally.

3 Experiments and Results

The training set includes 480 source-target input pairs, and the development set includes 120 source-target pairs. For each model, I run 100 epochs, where batches of samples are selected randomly at each epoch. Both the encoder and decoder models have 2 RNN layers, each with 100 hidden units. There is a dropout layer applied to the embedded inputs, with dropout rate of 0.1. I use Adam optimizer for both models. The learning rate of the encoder optimizer is 0.0001 and the learning rate of the decoder optimizer is 0.00005.

I experiment with different attention models: general, dot and concat, and different teacher forcing rate and batch size. The results are listed in Table 1.

Attention Mechanism As we can see from the results, the attention mechanism plays a crucial role on making this sequence-to-sequence model work practically. Without attention, the basic model can only achieve a denotation match rate of 0.075 and token-level accuracy of 0.590. Although the basic model as an auto-encoder can produce reasonable results, it performs poorly on the semantic parsing task. When the three types of attention are used, the denotation match rate increases to 0.425, 0.575 and 0.617 respectively. The token-level accuracy increases to 0.799, 0.846 and 0.852, respectively. Figure 1 shows the loss changes during training for the basic model and different attention types. The loss decreases significantly when attention is used. However, despite from the evaluation results on the development set, there is no significant difference between the three attention types during training time.

Teacher Forcing and Scheduling Figure 2 shows the loss changes during training for different teacher forcing rate settings. When the rate is 0, the decoder feeds its own generated output as

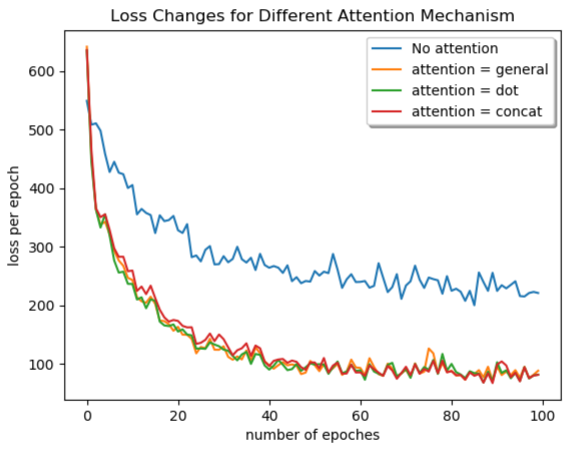

Figure 1: Train loss per epoch changes for different attention mechanism. The basic model has significantly higher loss than models with attention. The type of attention does not have large effect on the loss reduction.

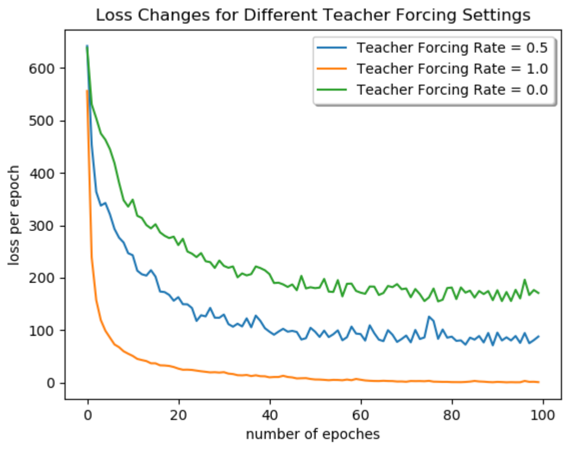

Figure 2: Train loss per epoch changes for different teacher forcing rate settings. The loss reduces fast and remains low when teacher forcing is used during the whole training. When teacher forcing is not used at all, the loss is much higher. When there is 50% chance to choose between the two settings, the loss is in the middle.

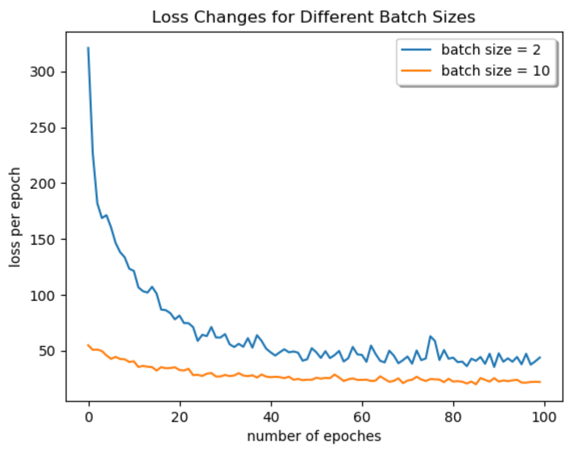

Figure 3: Average train loss per epoch changes for different batch sizes. The loss is the average loss per sentence. With larger batch size, the loss is consistently lower than that of smaller batch size.

the input of the next timestep. When the rate is 1, the decoder is constantly guided by the target values. Therefore the loss when rate is 1 is much lower than the loss when rate is 0. However, none of these approaches can achieve the best evaluation scores, because for the former case, the decoder output can wander too far if one output is wrong at certain step; for the latter, the decoder does not have the guiding "teacher" when evaluating on new data, hence loses the power. Combining these two methods hence gives the best results: 0.617 denotation match and 0.852 token-level accuracy.

Batch Size To compare the effects of different batch sizes on training loss, I divide the loss per epoch by the batch size to calculate the average loss per sentence, and plot it in Figure 3. When the batch size is large (10), the loss is consistently lower and has less variations. However, training the model with small batch size gives better performance on the development set: 0.508 denotation match and 0.820 token-level accuracy.

4 Conclusion

In this project, I implement the sequence-to-sequence model for solving the semantic parsing task. I also implement an attention mechanism to improve the accuracy. I use teacher forcing and batch training to facilitate the training process. I experiments with different parameter settings, and the best model has achieved 67.7% denotation match rate, 85.2% token-level accuracy and 60.0% exact match rate on the development set.

References

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025.

Spro. Practical pytorch: Translation with a sequence to sequence network and attention.

Yuk Wah Wong and Raymond J. Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 439–446, Stroudsburg, PA, USA. Association for Computational Linguistics.