CIS 5300: Assignment 4 [75 Points]

Yuling Liu Aman Kumar

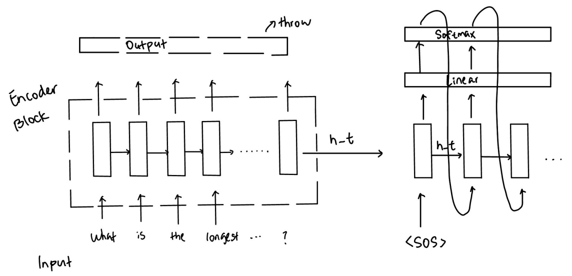
1 Introduction (10pt)

Our goal is to convert English sentences into queries so that we can execute against a knowledgebase. We used the standard GeoQuey dataset for the task. Here's an example of the task and dataset- if the question is "what is the density of texas?", and the result should be "_answer (NV , (_density (NV , V1) , _const (V0 , _stateid (texas))))". We trained the model with 480 examples in the train set, and evaluated the model on 120 examples in the development set.

To achieve the goal, we trained a sequence-to-sequence recurrent network model with attention, which teaches the model about structural properties of sentences.

2 Model Architecture (30pt)

The basic encoder-decoder model works by encoder taking in each item from the input sequence and generates a context vector and then the decoder takes in this context vector and outputs at each step, eventually generating a whole output sequence. In our case, both encoder and decoder modules use LSTM as the RNN unit. See the following graph for the architecture of our basic model.



For the training part, the encoder module generates an output of a batch of input sequences (example: 2 sequences), then in order to use the encoder output in the decoder module, we iterate over the batch of the input sequences, and access the encoder output for a single sequence for which we want to get the decoder output sequence. We use the this encoder output in every step for getting the decoder output word. The decoding part in training starts with SOS token and then we obtain the next token and iterate till end of the output sequence length. The predicted token and original token are used to get the loss for that step. This loss is also added to a variable called "batch loss" that calculates the loss for the input batch. The loss function used is negative log likelihood loss.

The decoder attention module takes arguments embedded input, hidden state, encoder output for that complete sequence and encoder context mask. The embedded input starts with SOS token and then the predicted token is used as the embedded input for next iteration in case of decode/testing. The initial hidden state is the encoder state output h_t. This initial embedding and initial hidden state is input to the LSTM module which generates an output and a new hidden state. We will ignore this output for implementing attention.
Now we use this new hidden state and encoder output for this sequence and get a score and then take a softmax to get the probability values and then multiply these values with encoder output to get a context vector.

$$e_{ij} = s_{i-1}^T h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

Now this context vector c_i is concatenated with the new hidden state and given to a feed forward network. Then the output of this is the of size of output dictionary. We then use a log softmax to get the log probability values for the output.

$$o_t = feedforward([s_{t-1}; c_t])$$

$$logprob = logsoftmax(o_t)$$

In the decoding function, we take the argmax of this output vector to get the output token indices.

Attention model improves the performance as the model is focused on the relevant parts that are needed from the input sequence when generating output at each step. This solves the problem of directly taking the encoded output and passing it to decoder, because in this case the encoded output has more information/focus of the last few tokens in the input sequence, which may not be prudent to use in this question answer task. As different areas of the input question should be taken into consideration in order to get the correct answer. And this is solved by attention, that is why the output is much better.

3 Model Optimization (10pt)

We have used Adam optimizer for our model. First we vary on the learning rate, so we keep epochs = 10, hidden size = 200, and batch size = 2. We get the following results:

LR	Exact	Token	Denotation
1e-3	0.208	0.738	0.242
1e-2	0.150	0.643	0.242

Now we determined the learning rate to be 0.001. Training the model at different epochs results in the following:

Epoch	Exact	Token	Denotation
10	0.208	0.738	0.242
20	0.383	0.808	0.458
30	0.408	0.818	0.458

We tried epochs of 10, 20, and 30, and 30 yields the best results. Then we varied different batch sizes and obtained following result:

Batch	Exact	Token	Denotation
1	0.492	0.783	0.550
2	0.408	0.818	0.458
4	0.375	0.817	0.417

Finally, we have learning rate = 0.001, batch size = 1, epoch = 30 and lastly we determined the number of hidden layers for crossing the exact match score of 0.45.

Hidden	Exact	Token	Denotation
200	0.492	0.783	0.550
300	0.517	0.800	0.558

So for a hidden layer size of 300 we get the best exact match score which crosses the exact match score of 0.45 for the test data.

4 Output Results (10pt)

The table below is a summary of our results from the development set. The advanced model with attention yields the best results.

	Exact	Token	Denotation
Nearest	0.125	0.697	0.200
Basic	0.025	0.592	0.075
Advanced	0.517	0.800	0.558

5 Error Analysis (10pt)

Below is a table summary of failure cases and reasons

Class	Example
Wrong number	what is the population of sacramento? False: [275741count] == [904078count]
Join failed syntactically	how long is rio grande? False: [3033000length] == [Join failed syntactically]
Example FAILED TO EXECUTE	how many rivers are there in us ? False: [46count] == Example FAILED TO EXECUTE
Wrong answer	what are the rivers in the state of indiana? False: [wabash:riverid/2],[ohio:riverid/2] == [colorado:riverid/2]
Example FAILED TO PARSE	what is the population of the state with the largest area ? False: [401800count] == Example FAILED TO PARSE
Answer not found	in what state is mount mckinley ? False: [alaska:stateid/2] == {}

In addition, there are total 6 syntax errors, 13 executor errors in the development dataset.

6 Conclusion (5pt)

The sequence-to-sequence model with attention yields the best result (token-level accuracy of 0.800). While the token-level accuracy is not low, the exact logical form matches are not good enough. The model still produces different types of errors, such as failure to execute, and parse.

Given more data, this attention sequence to sequence model can get much better in getting the inference. Some other work that can be done on this model is for example: adding more LSTM layers, using different scoring function, regularization mechanism like a Dropout layer. These techniques can be used to enhance the performance of the model.