Semantic Parsing with Encoder-Decoder Models

Shivam Garg
shivgarg@cs.utexas.edu

1 Introduction

In this project sequence to sequence models have been explored for semantic parsing. The task of semantic parsing involves converting a natural language query to a form which encodes the structure of the query and the parsed form can be used to perform further analysis. The problem is an important part of question answering, machine translation and information retrieval. In the current project, natural language queries are converted to prolog formulas and the formula is solved using a knowledge base. The dataset used for analysis is the Geoquery dataset (Zelle). Seq2seq models with LSTM cells and attention have been used to solve the problem at hand. Beam search and batch processing have been used to improve the performance of the system both in terms of accuracy and training speed.

2 Sequence to Sequence Models

Sequence to sequence models have proved to be very useful for various tasks in NLP. These models translate one sequence to tokens to another sequence of tokens. The framework consists of an encoder and decoder. Encoder encodes the sentence into a single vector. The output of the encoder is given as input to decoder which in turn produces the target sequence. This framework can be used to generate arbitrary translations of input sequence of tokens. In the current assignment, the encoder and decoder are made of LSTM cells.
LSTM is a variant of recurrent neural networks. They have been designed to be good at capturing long term dependencies in a sentence. The LSTM cell consists of four gates:- forget gate, input gate, candidate gate and output gate. These gates work in tandem to filter and select information which is stored in the cell and hidden state.
The seq2seq model with single LSTM cell for encoder and decoder was trained end to end. The words were embedded into an embeddingwhich was initialised randomly and was trained alongside the seq2seq model. For decoder, teacher forcing was used for training. The embedding for both input and output tokens was 300 dimensional. The hidden state of LSTM was 500 dimensional. Adam optimiser with learning rate of 0.001 and batch size of 4 was used to optimise the model. A fully connected layer was used to produce likelihood score over the output vocabulary at each position in the output. The model was trained for 20 epochs. The accuracy numbers are listed in Table 1. Some example outputs of the model on the dev set are shown in Table 2. A comparison is made between the outputs of LSTM encoder-decoder and LSTM+Attention encoder-decoder.

3 Attention in seq2seq

One of the major drawback in the architecture of seq2seq models is that the whole sentence has to be encoded into a single vector and the decoder has to produce the output based just the encoded vector. The entities which are not frequent in the training set or are present in context of more popular entities are supressed and are not well learnt by both encoder and decoder. Moreover, although LSTM have been designed to factor in the long term dependecies , in practice they are not able to capture dependencies in sentences with length greater than 20. They tend to forget the semantics of the initial part of sentence, leading to a biased encoded vector which results in poor performance. To overcome the above limitations, attention mechanism is used . In attention, apart from the encoded vector, the decoder has access to the hidden states produced at each position of the input. The attention module uses the current decoder hidden state to calculate the relevance of input hidden states. One of the ways a propsed by (Luong et al., 2015) is to calculate the cosine similarity of two vectors. Another way is to learn a ma-

Model	Exact logical form	Denotation	Token-level
LSTM	0.358	0.383	0.812
LSTM+Beam Search	0.375	0.442	0.812
LSTM+Attention	0.500	0.542	0.799
LSTM+Attention+Beam Search	**0.550**	**0.600**	**0.819**

Table 1: Accuracy Stats on dev set

Input	Ground Truth	LSTM	LSTM+Attention
"what is the longest river flowing through new york ?"	_answer(A, _longest(A, (_river(A), _traverse(A, B), _const (B, _stateid ('new york')))))	_answer (NV , _longest (V0 , (_river (V0), \+ (_traverse (V0 , NV) , _const (V0 , _stateid (**'new mexico '**)))))	_answer(NV, _longest (V0, (_river(V0), _traverse(V0, NV), _const(V0, _stateid (**'new york'**)))))
"where is austin?"	_answer (NV , (_loc (NV , V1) ,_const (V0, _cityid (austin,_))))	_answer (NV , (_loc (NV , V1) ,_const (V0, _cityid (**springfield**,_))))	_answer (NV , (_loc (NV , V1) ,_const (V0, _cityid (**austin**,_))))
"which states border iowa ?"	_answer (NV , (_state (V0), _next_to (V0, NV) , _const (V0, _stateid (iowa))))	_answer (NV , (_state (V0), _next_to (V0, NV) , _const (V0, _stateid (iowa))))	_answer (NV , (_state (V0), _next_to (V0, NV) , _const (V0, _stateid (iowa))))

Table 2: Examples of model output

trix which is then used to apply an affine transformation to the decoder hidden state and this transformed hidden state is used to calculate similarity with the encoder hidden states. After determining the similarity, the hidden vectors from encoder are summed with weights of each hidden vector being equal to the softmax probability of the similarities calculated for each vector. The weighted sum vector and the decoder hidden vector are then concatenated, which is then passed to the fully connected layer for prediction. In the current implementation, the vanilla dot product attention function is used. Rest of the implementation details were same as the network trained without attention.

4 Extension

Two extensions have been focused in this project. First one is based on beam search and second one is batch training of decoder for faster training time.

4.1 Beam Search

Current inference procedure involves greedy selection at each step of output sequence generation. This sometimes leads to suboptimal output as greedy selection can lead to worse outputs in future. To solve this exactly, Viterbi algorithm can be used which considers all posssibilites at each time step. But there are two major drawbacks of using Viterbi algorithm. Firstly, the computational complexity is proportional to vocabulary size, which is typically huge, leading to very slow inference procedure. Secondly, seq2seq models can produce variable length sequences for the same input so some book keeping is required to adapt Viterbi to variable length outputs.
Beam search is an approximation to Viterbi algorithm where instead of all possible outputs, top k scoring outputs for a particular state are stored. All these partial solutions are used to expand upon in the next step. In the current implementation a beam of size 5 is used and the average of log probability of generating the sequence is used to rank the solutions.

4.2 Batch Training of Decoder

A simple way to implement decoder is to pass a single token to decoder, process the output of de-

Model	Runtime/epoch(in secs)
LSTM-Non-batched	84.3
LSTM+Attention-Non-batched	90.5
LSTM+Batched	16.4
LSTM+Attention-Batched	16.8

Table 3: Training time for 1 epoch(in secs)

coder and add the loss to the batch loss.This process needs to be repeated for all tokens in the training batch. This approach is slow since it does not exploit the parallel algorithms implemented in pytorch which process all examples in a batch together.
Another way to implement is to pass the whole batch together and perform computation in one go. The batch is passed as a packed padded sequence to the decoder, which runs the lstm cell over all the examples in the batch simultaneously. As the input sequence to the decoder is padded with the PAD symbol,the loss of the model due to the padded input is ignored. In the loss function, a mask was computed which specified the tokens whose loss should be used for backpropagation. The speedup obtained on CPU are listed in Table 3. The batched version of the code was 5-6 times faster than the non-batched ones.

5 Conclusion

Seq2seq models perform decently well on the semantic parsing task as compared to traditional parsing based methods. These methods did not require any hand coded rules nor any special architecture. Simple seq2seq models applicable to various tasks in NLP worked well. The models without attention suffer with low denotational accuracy. On analysis, as discussed in class, many prolog formula are structurally correct, but the entity name is incorrect whch leads to poor denotational performance. With attention the denotational accuracy jumps by approximately 16%. Majority of the changes happen in the examples in which wrong entity was picked by LSTM encoder decoder. Moreover, the token level accuracy in attention case is in the same region as of normal case, further giving evidence that attention is mostly correcting the entities and not the structure of the prolog formula.

Beam search is helpful in increasing the performance of the system. A jump of 6% is observed in denotational accuracy which is 16% and 10% relative improvement in normal lstm encoder-decoder and lstm encoder-decoder with attention respectively. Batching training process gives 5-6x speedups in training time which underlines the importance of properly implementing the models to gain full benefit of optimised libraries routines.

References

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

John M Zelle. Learning to parse database queries using inductive logic programming.